Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Enterprise Applications and Services

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

National Chengchi University

企業應用程式

- Enterprise applications
  - Payroll
  - Patient records
  - Shipping tracking
  - Cost analysis
  - Credit scoring
  - Insurance
  - Supply chain
  - Accounting
  - Customer service
  - Foreign exchange trading
- NOT Enterprise applications
  - Automobile fuel injection,
  - Word processors,
  - Elevator controllers,
  - Chemical plant controllers,
  - Telephone switches,
  - Operating systems,
  - Compilers,
  - Games

企業應用程式特性

- Persistent data
  - 企業系統主要用來保存、維護、操作企業重要資訊
  - 這些資訊甚至會存活比程式、機器更久

機器會汰舊換新
程式也會逐漸被淘汰

- A lot of data
  - 中型系統: 1G 以上資料需要保存
  - 一般存於RDBMS

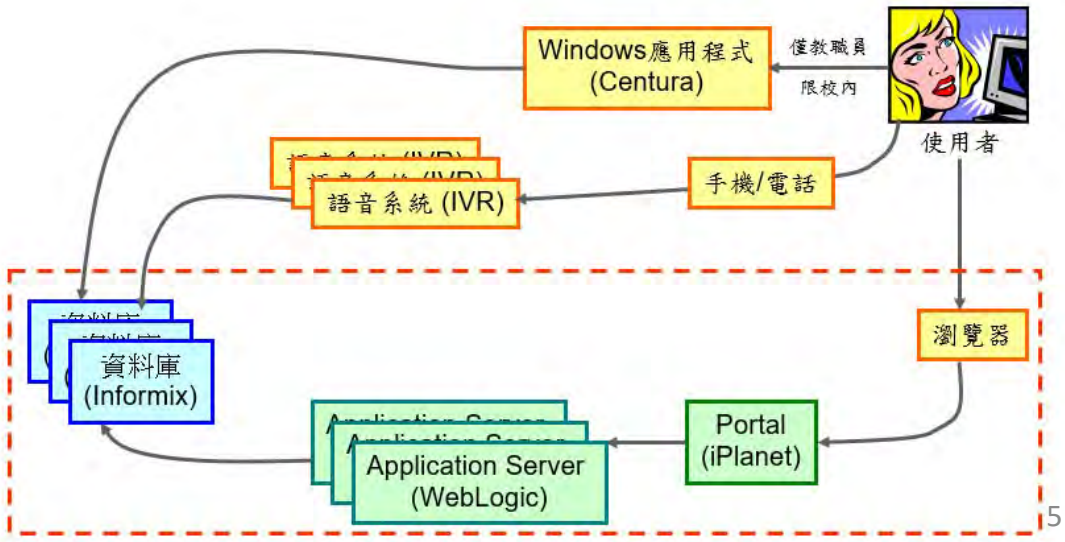
關聯式的傳統資料庫，NoSQL 通常輔助用

  - 更早期系統存於IBM VSAM/ISAM
    - Indexed Sequential Access Method
    - Virtual Storage Access Method

企業應用程式特性

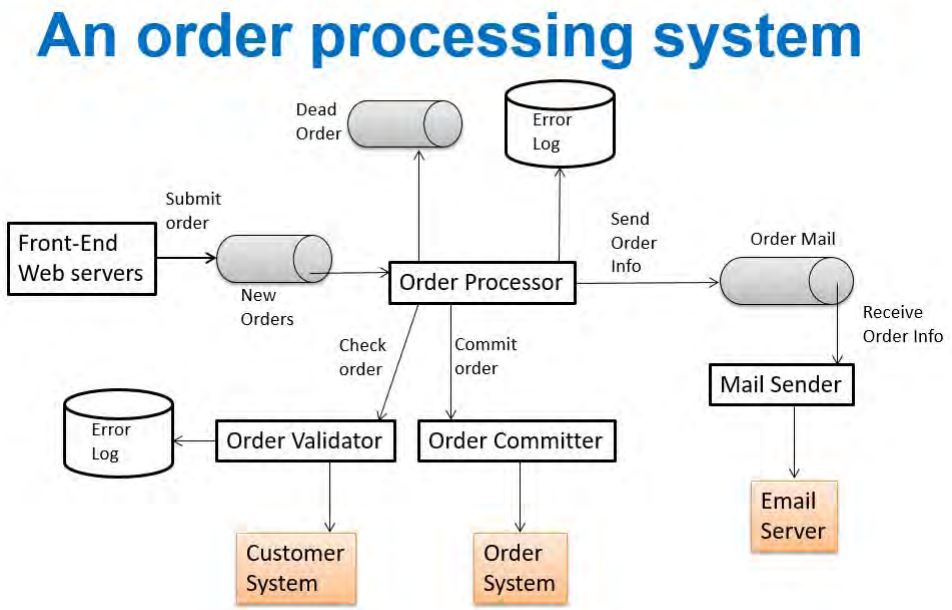
- Access data concurrently
  - Race Condition: 同時寫入同筆資料會出現問題
  - Transactions概念變重要
- User interface screens
  - Web/Client-Server/IVR

政大校務行政系統

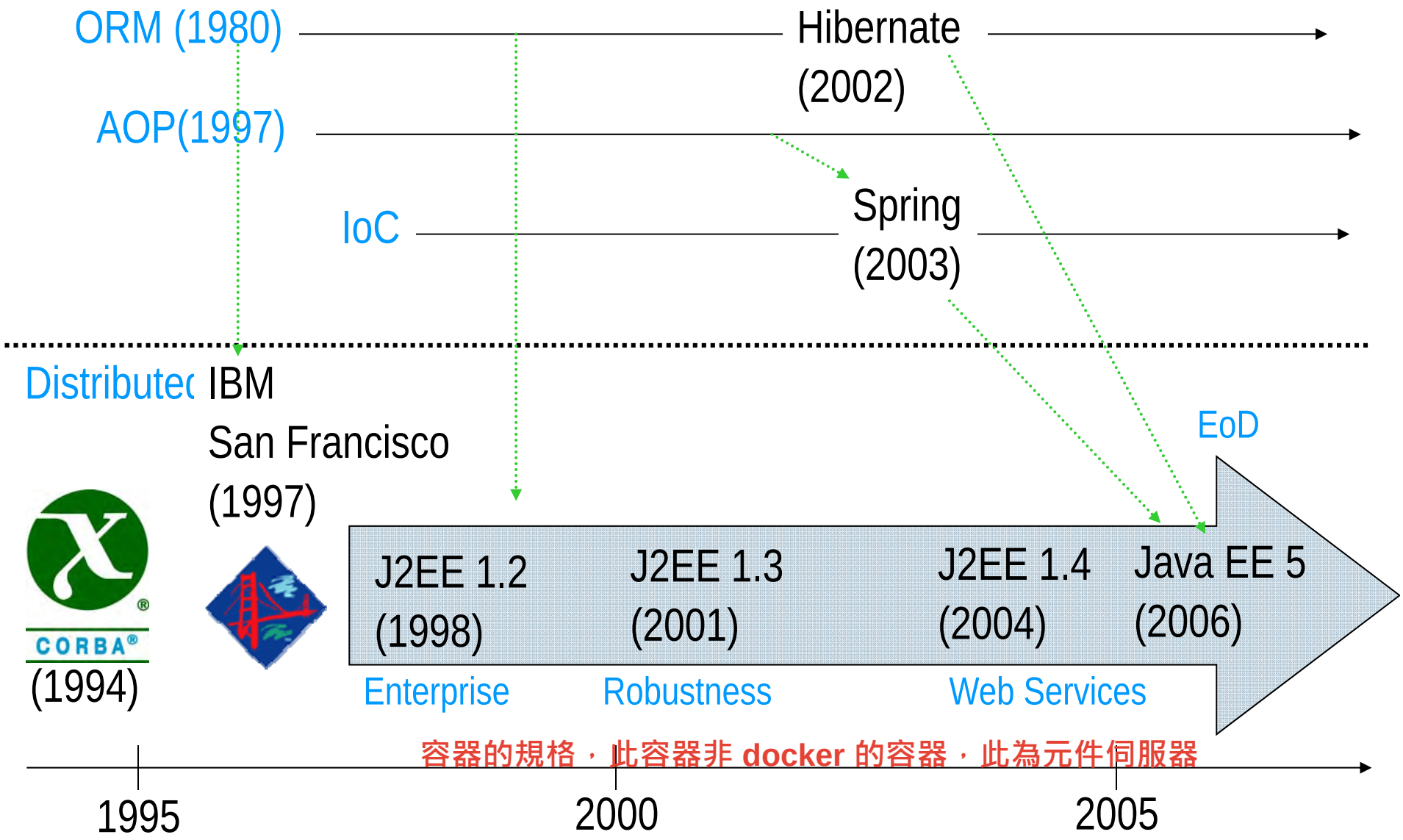


企業應用程式特性

- Integrate with other enterprise applications
  - Enterprise applications rarely live on an island
  - Typically via COBOL, Web Services, or Messaging Systems



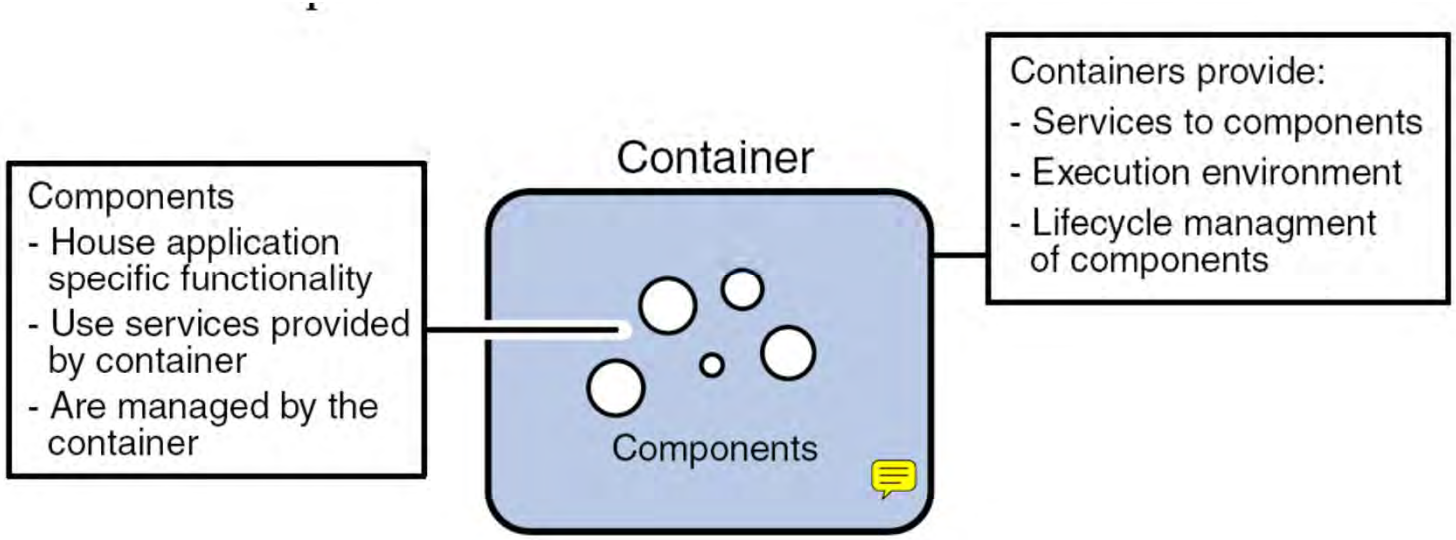
Java 企業端技術的進化



元件

容器

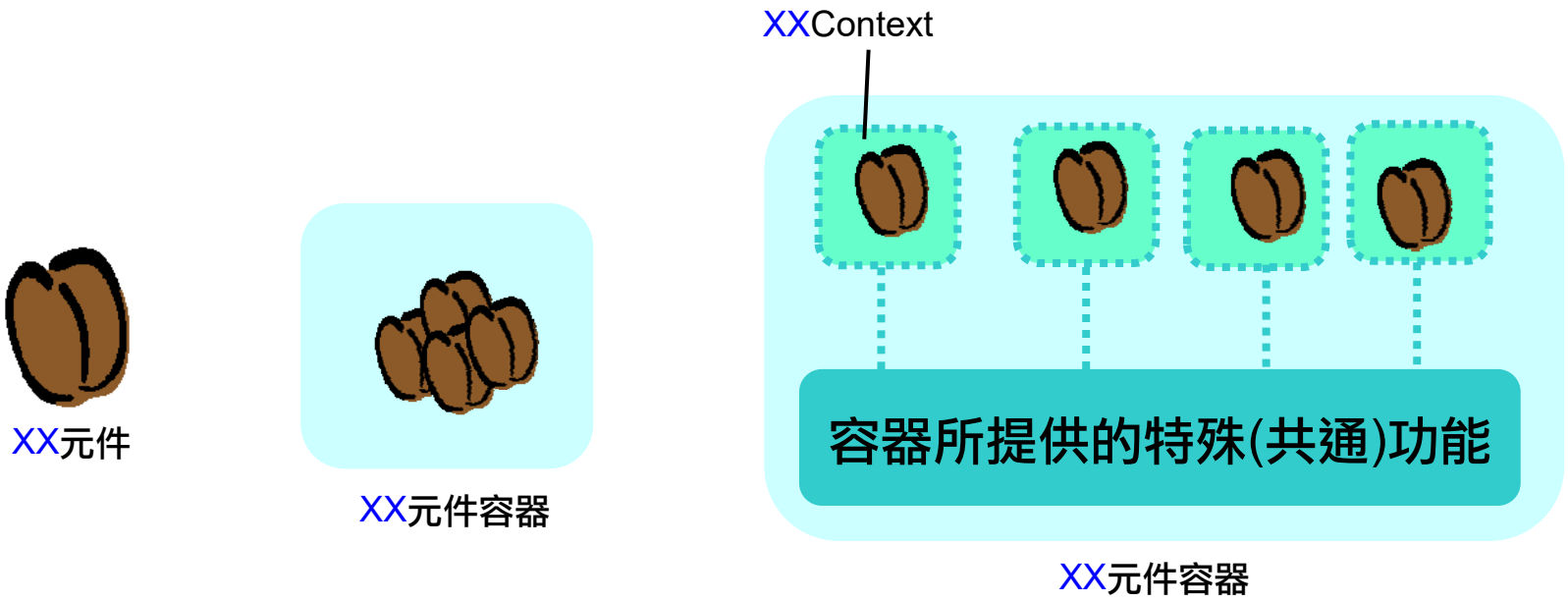
容器/元件架構



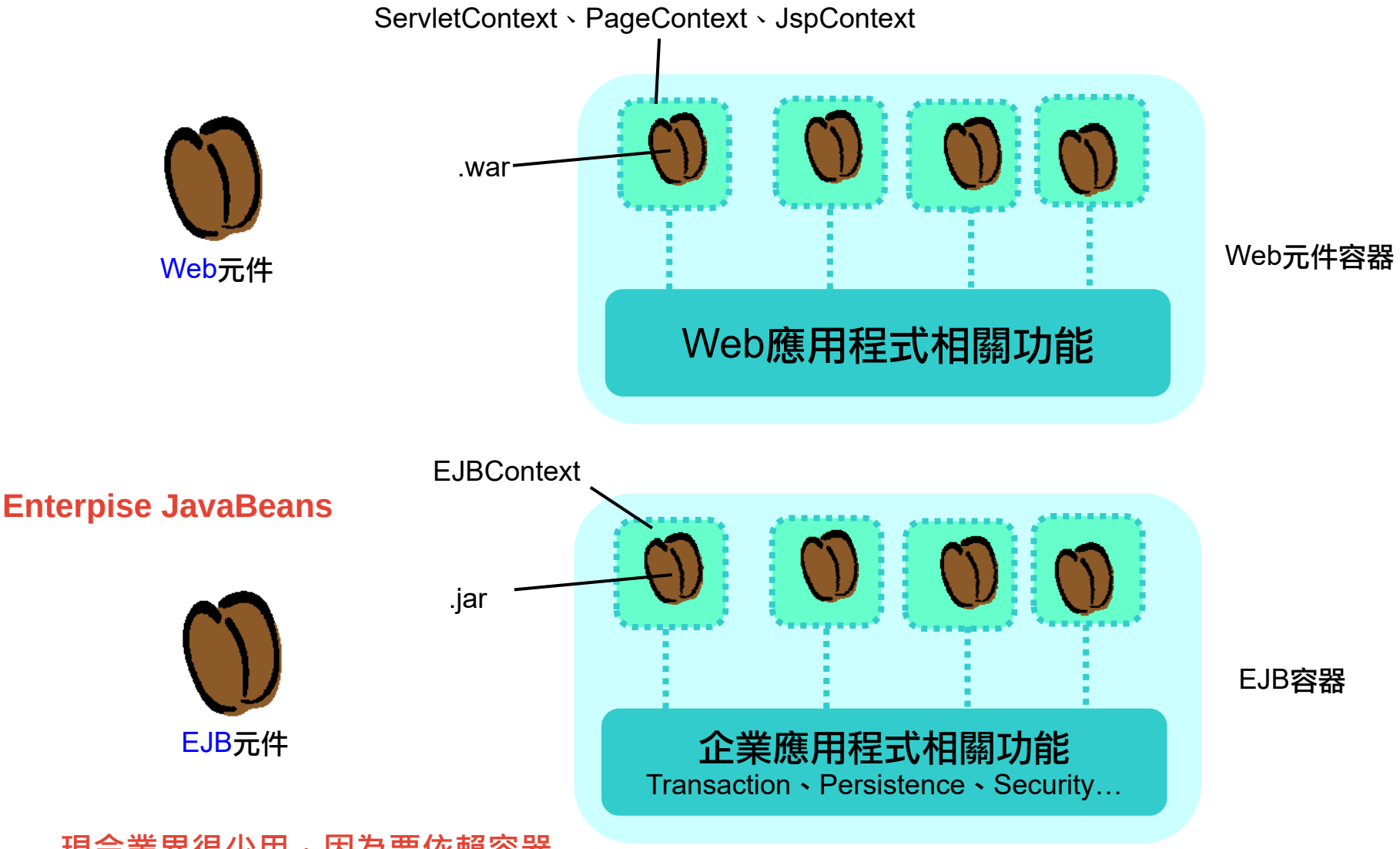
元件/容器溝通的媒介稱為Context

透過其來進行訊息交換

元件-容器模型

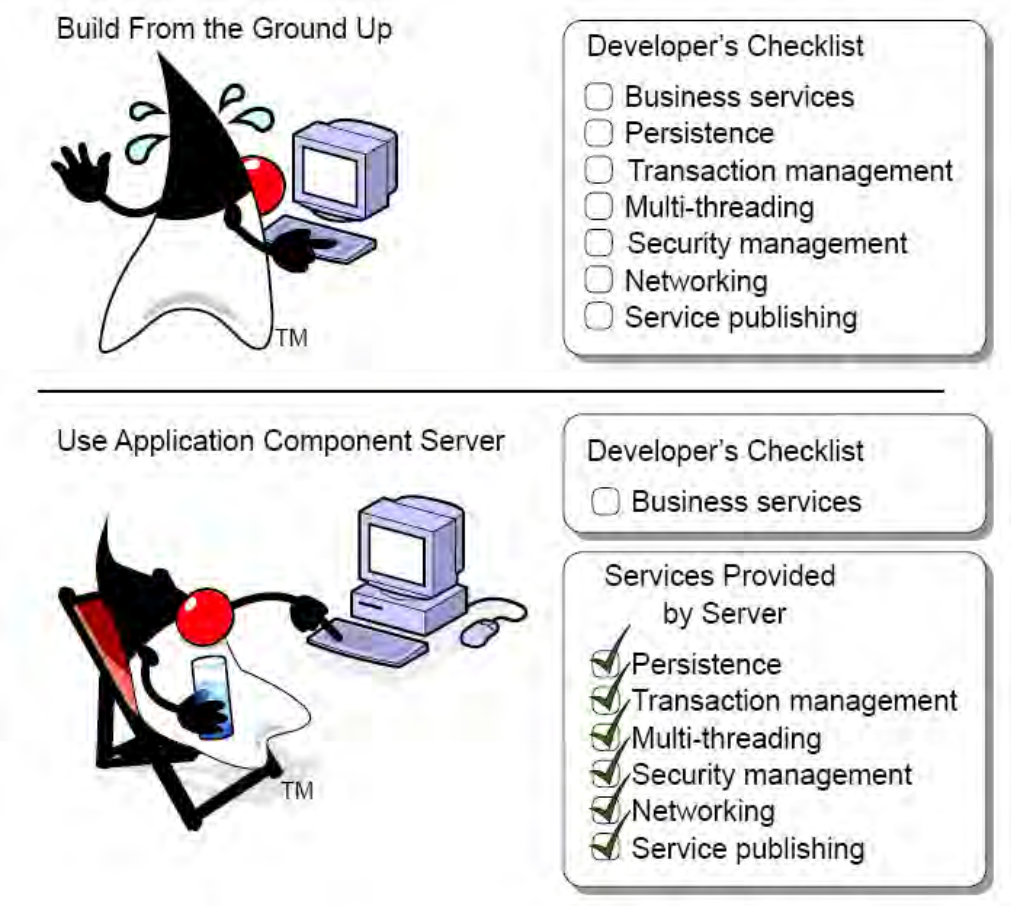


常見的元件模型



現今業界很少用，因為要依賴容器

和傳統開發方式比較



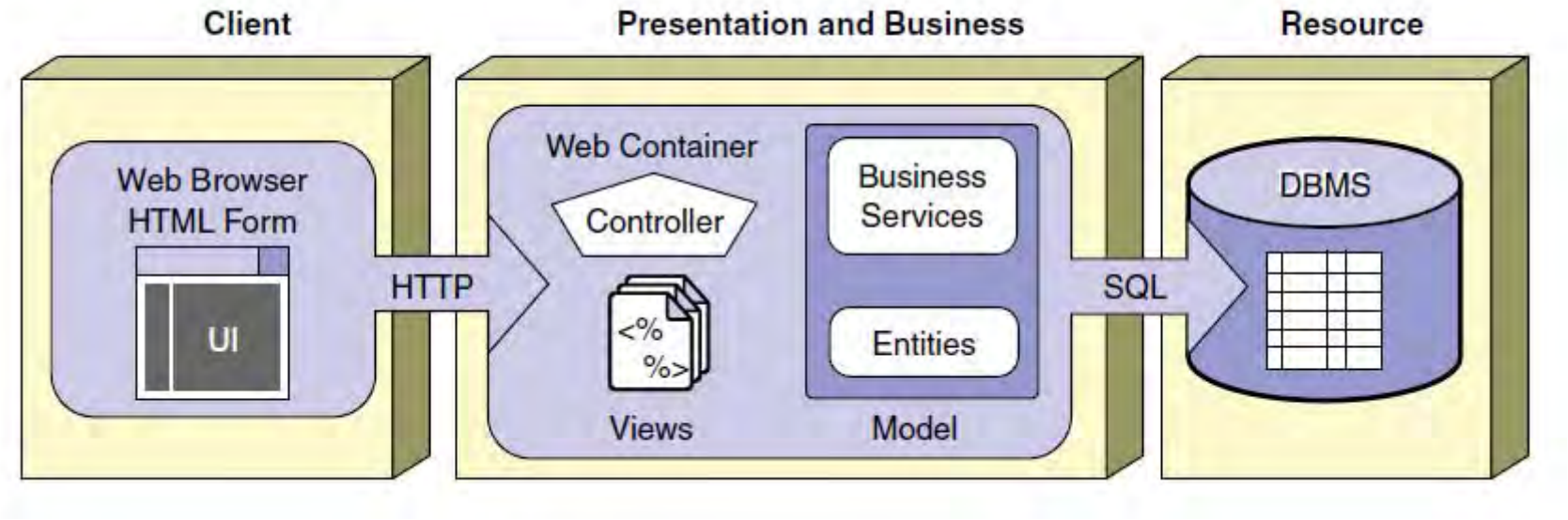
開發人員可免除開發Cross cutting concerns的麻煩

現今較少用

企業應用程式架構

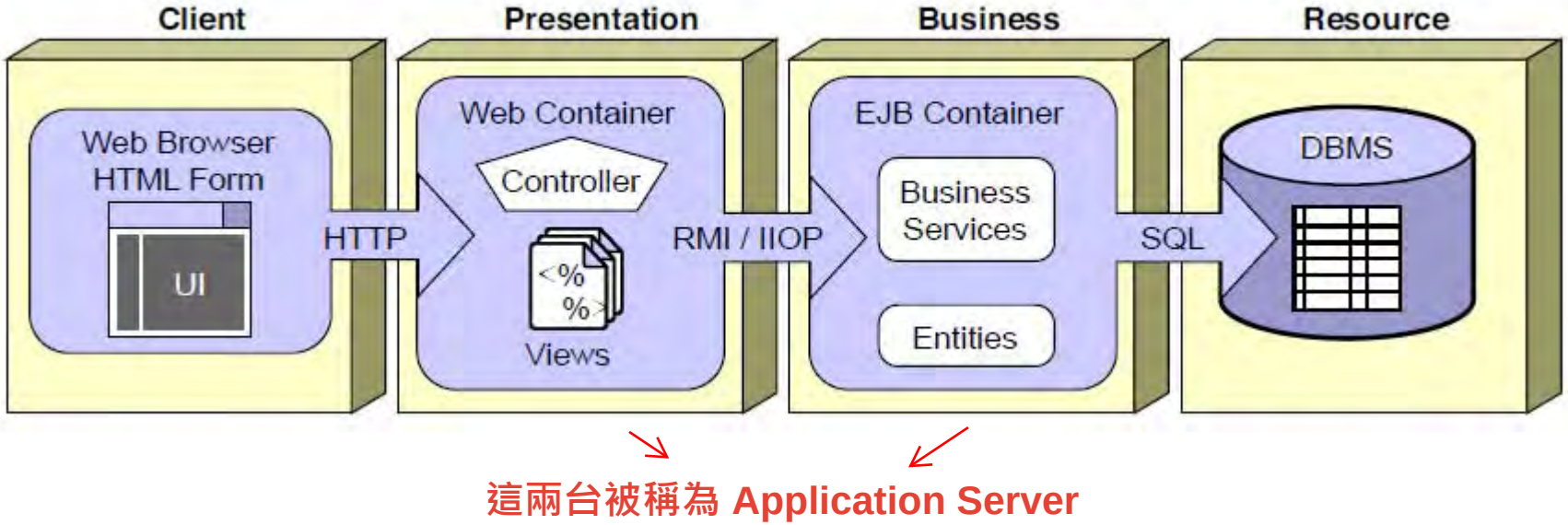
- Web centric

DAO(Data Access Object)
在跟資料庫交流的東西



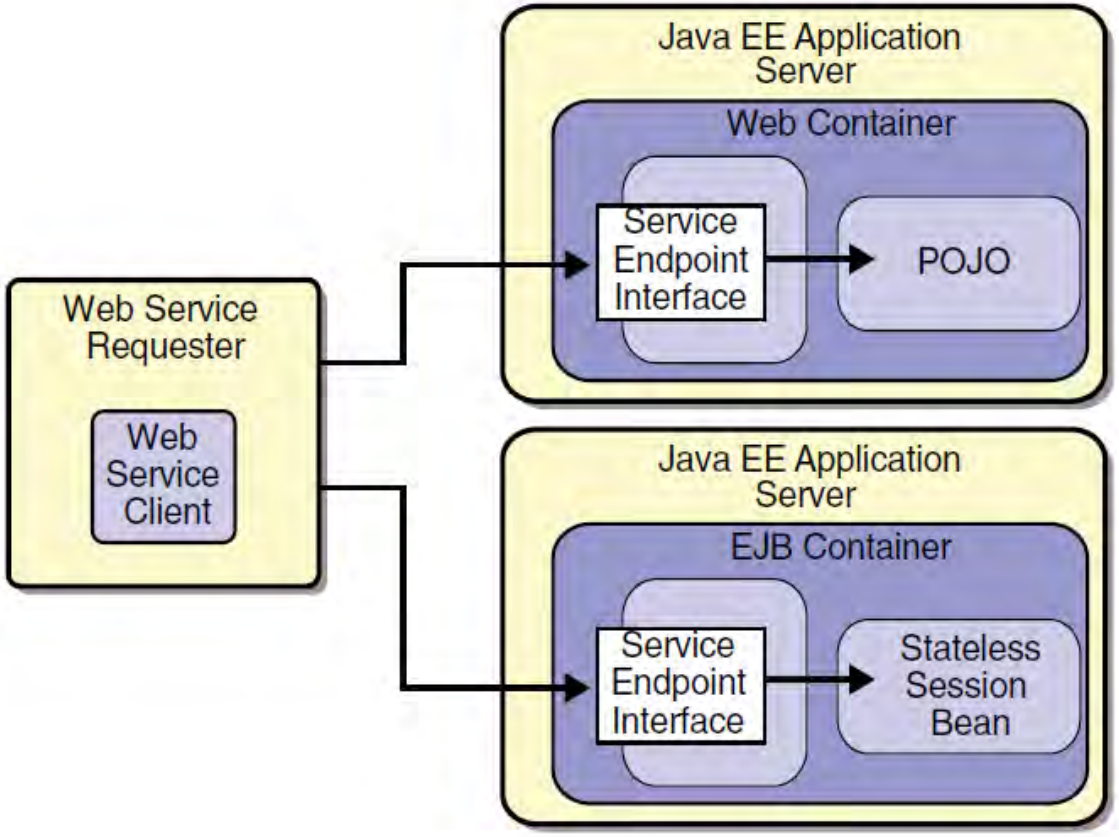
企業應用程式架構

- Middleware centric
  - Ex: Java EE containers



企業應用程式架構

- Service Oriented

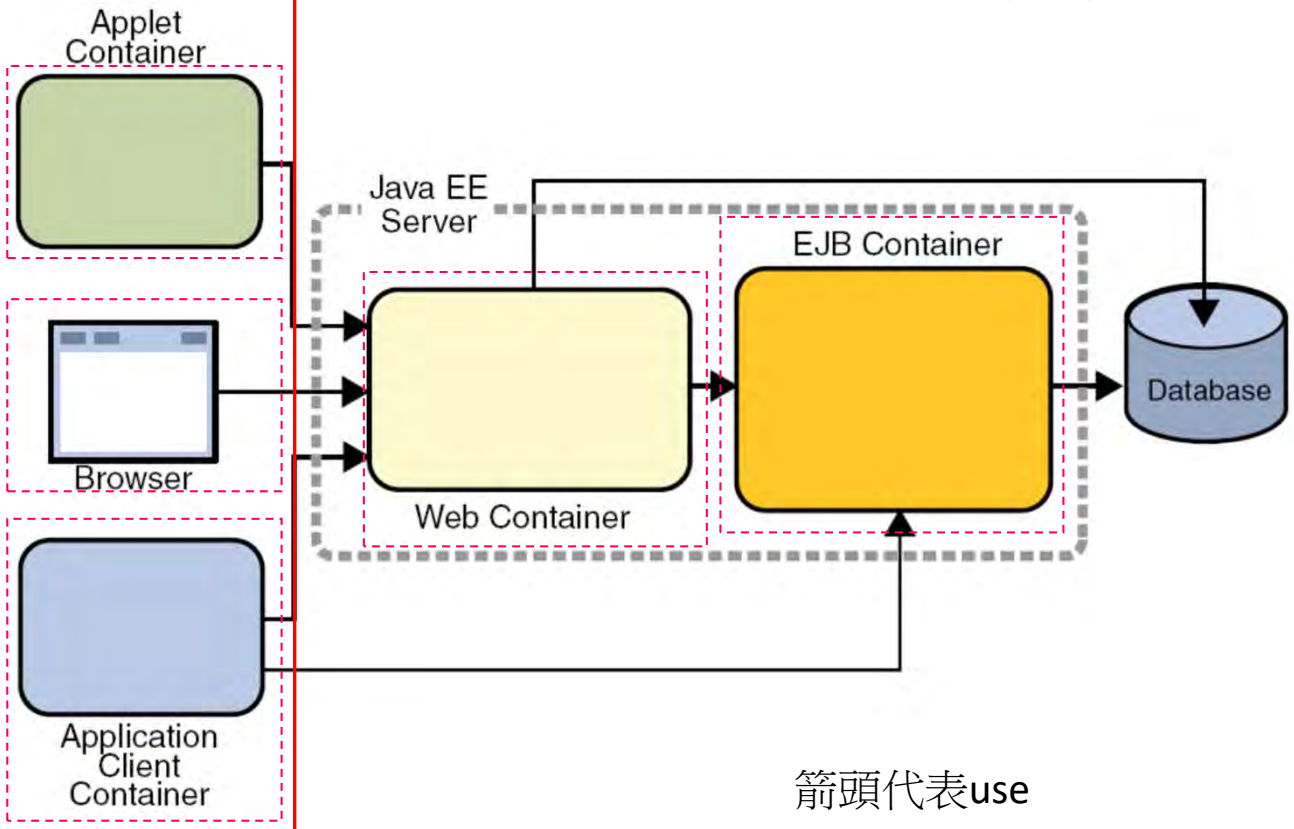


Front End

Java EE 的容器

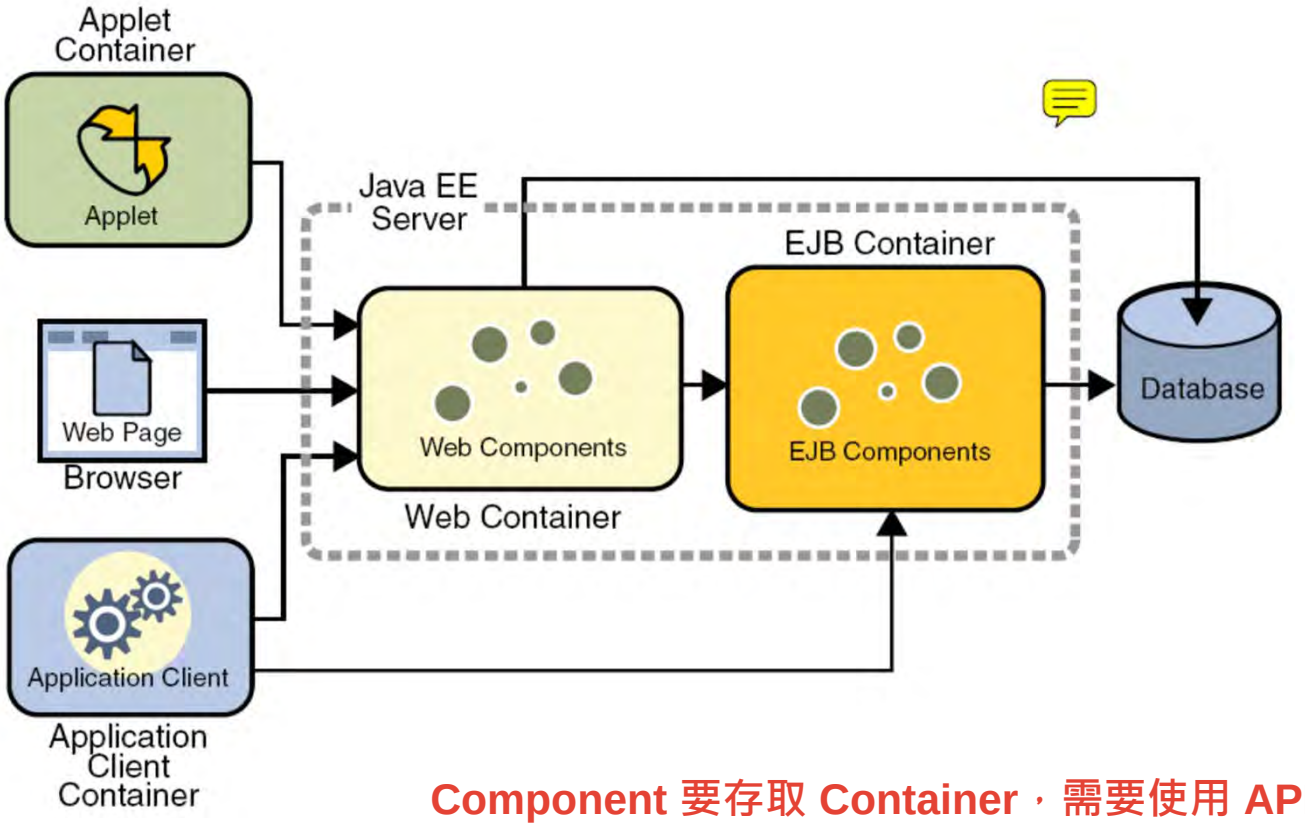
Back End

這些容器的元件各是什麼?



箭頭代表use

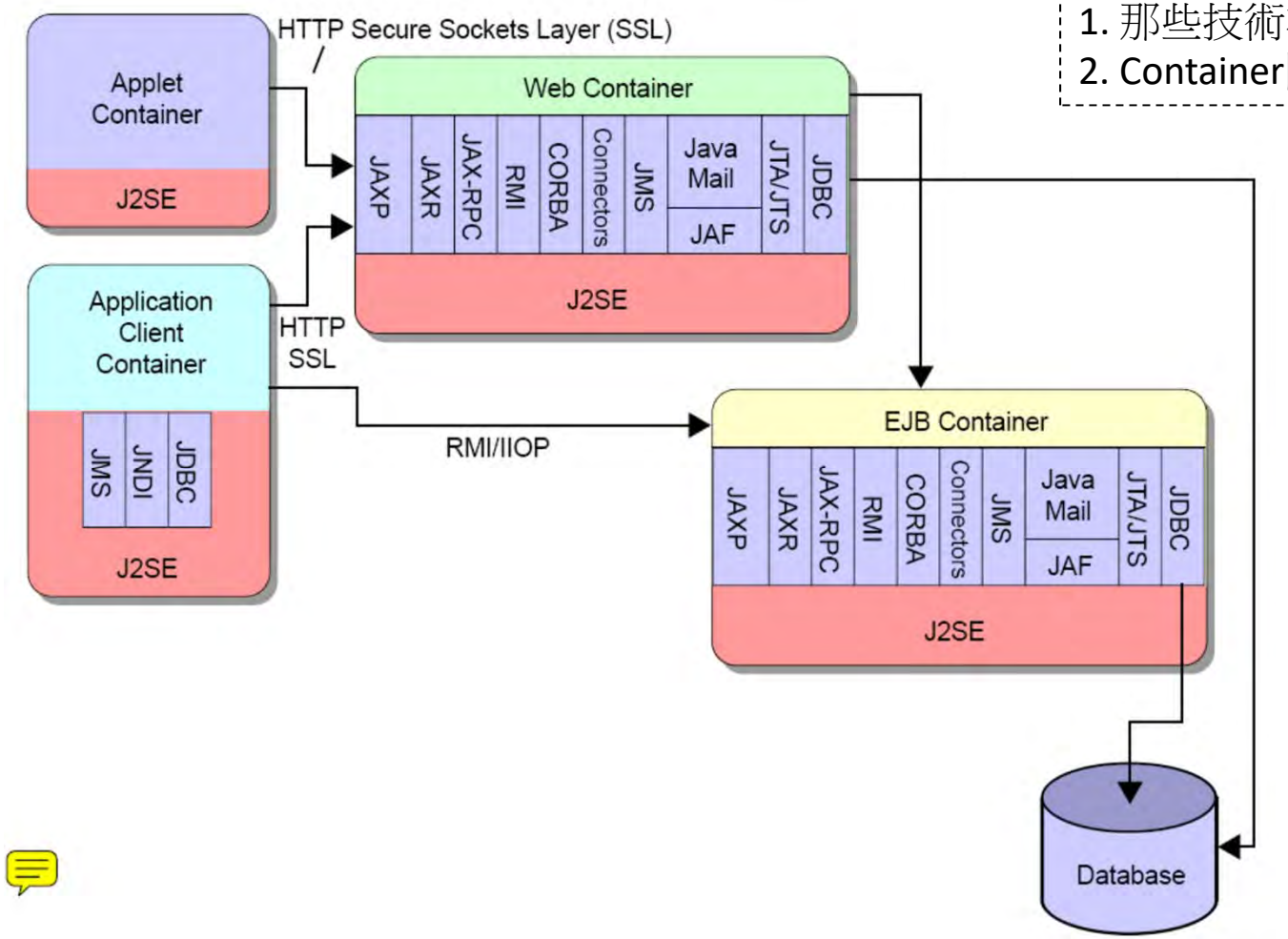
Java EE 的元件



Component 要存取 Container，需要使用 API

API Services

觀察重點:
1. 那些技術在那些Container提供
2. Container間溝通的協定

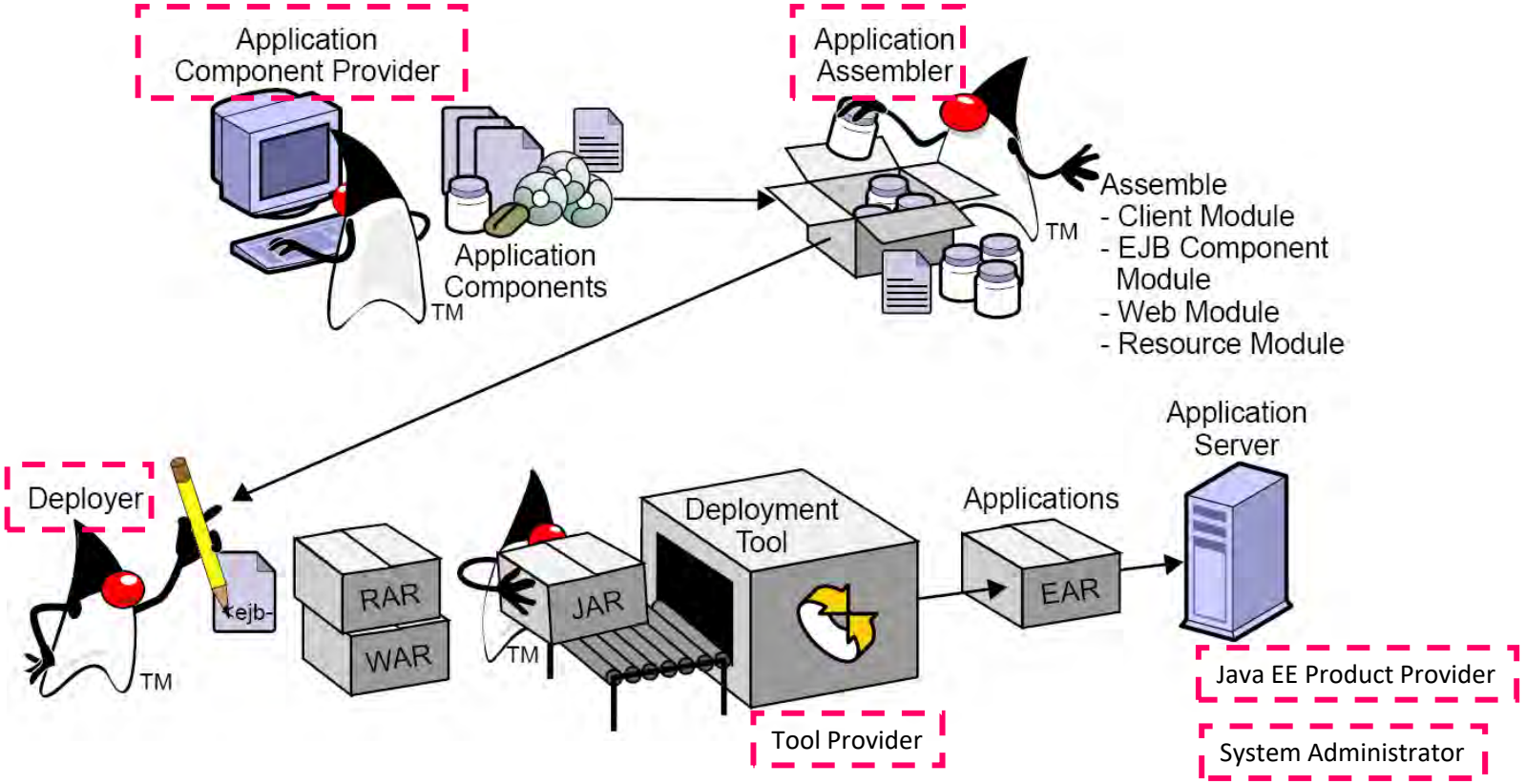


Application Creation Process

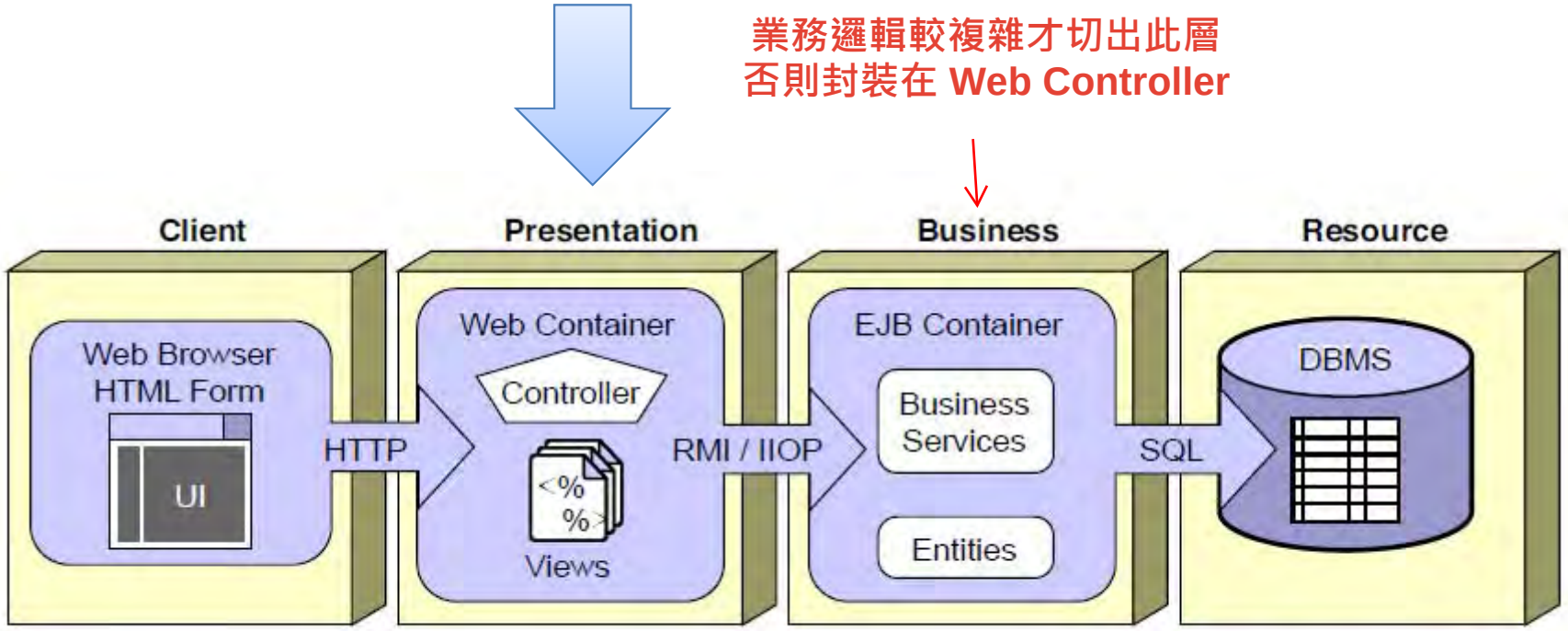
- 分析階段
  - 使用OO/UML技術分析Functional Requirement
    - 分析Logic
    - 分析Data
  - 分析Non-functional requirement
- 實作階段
  - Data→Entity class
  - Logic→SessionBean
  - Async→MDB
  - Client
- 佈署階段
  - Assemble / Packaging / Deploy

哪些功能？
功能品質的程度？

Java EE Roles (EO 1.4)

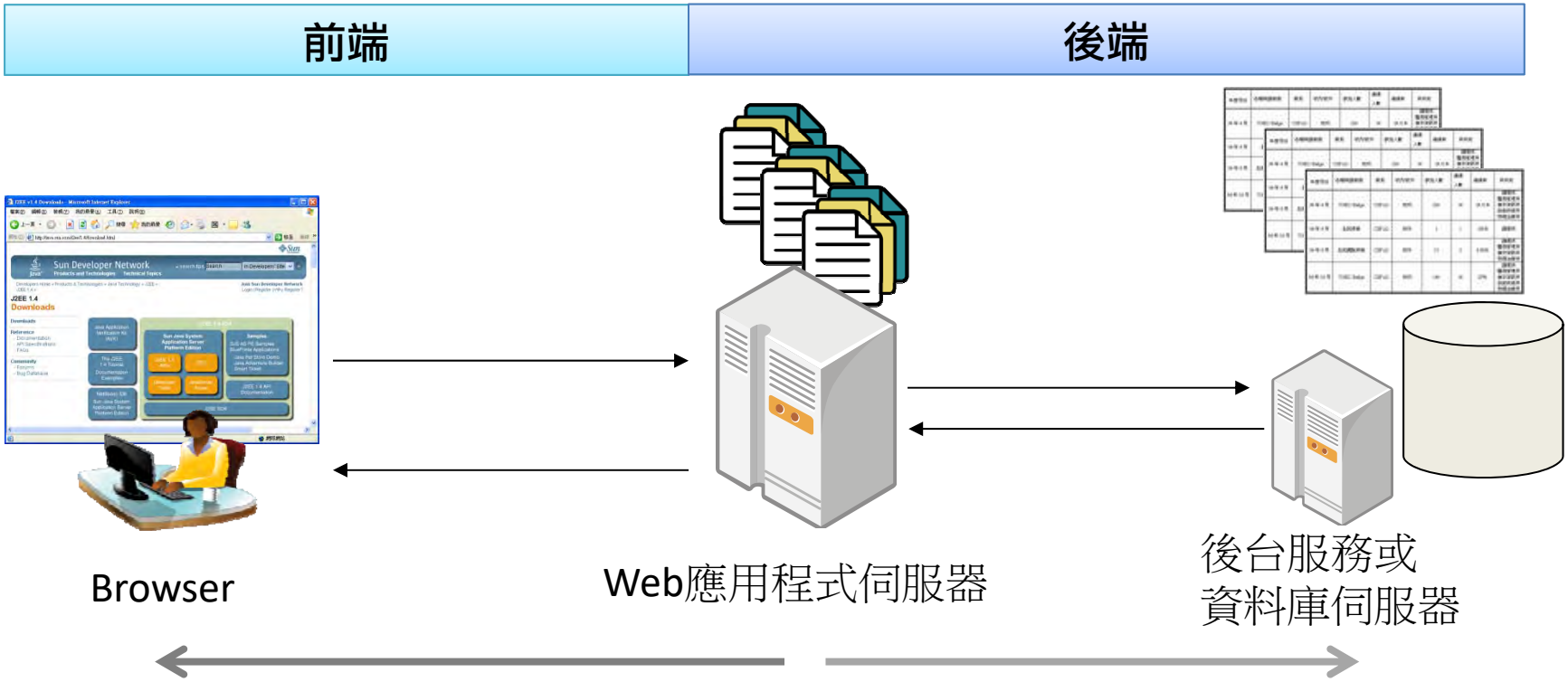


Web應用程式架構

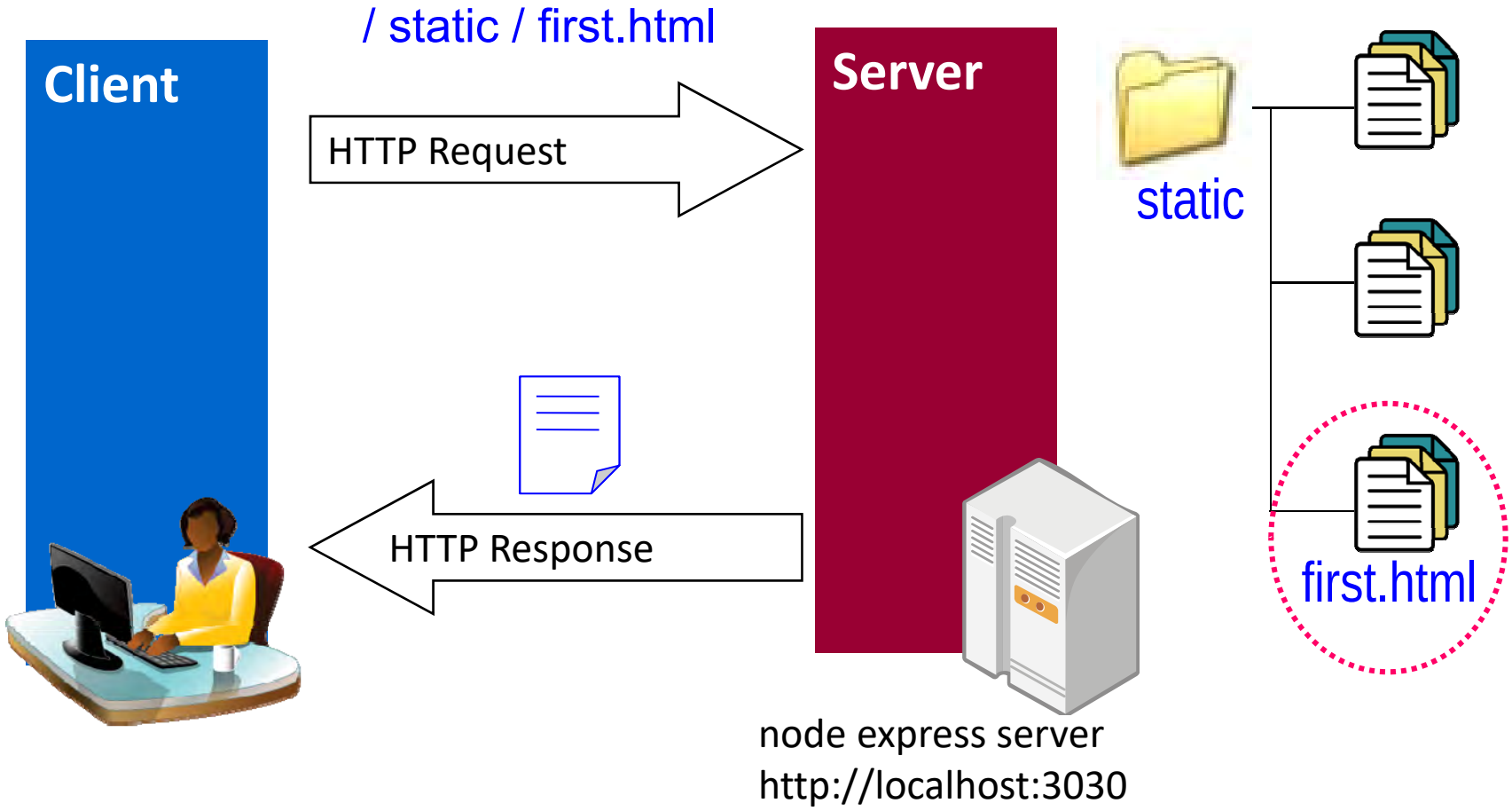

業務邏輯較複雜才切出此層
否則封裝在 Web Controller
Client
Presentation
Business
Resource
Web Browser
HTML Form
UI
HTTP
Web Container
Controller
<%
%>
Views
RMI / IIOP
EJB Container
Business
Services
Entities
SQL
DBMS

Web應用程式架構

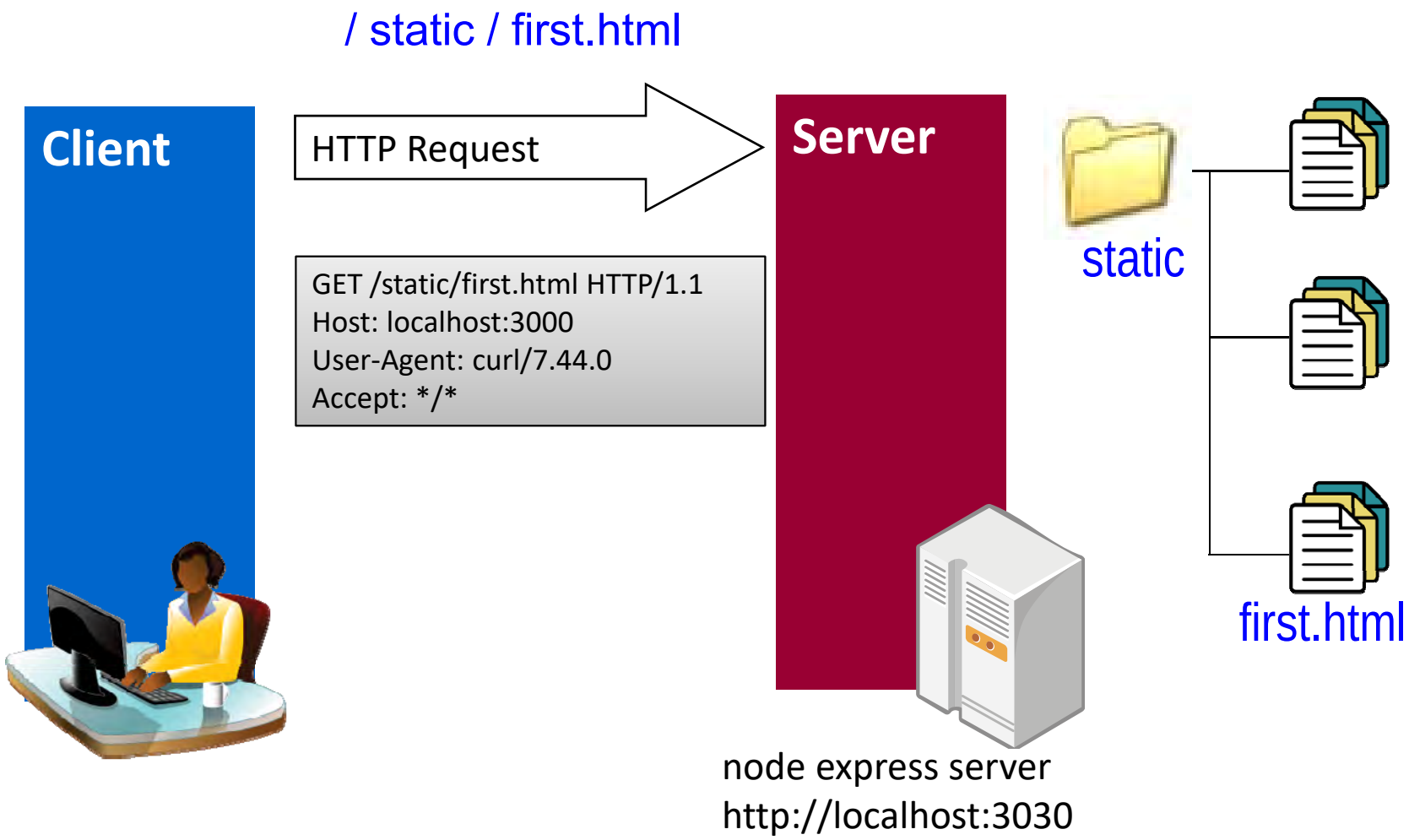
前端互動、後端資料庫互動



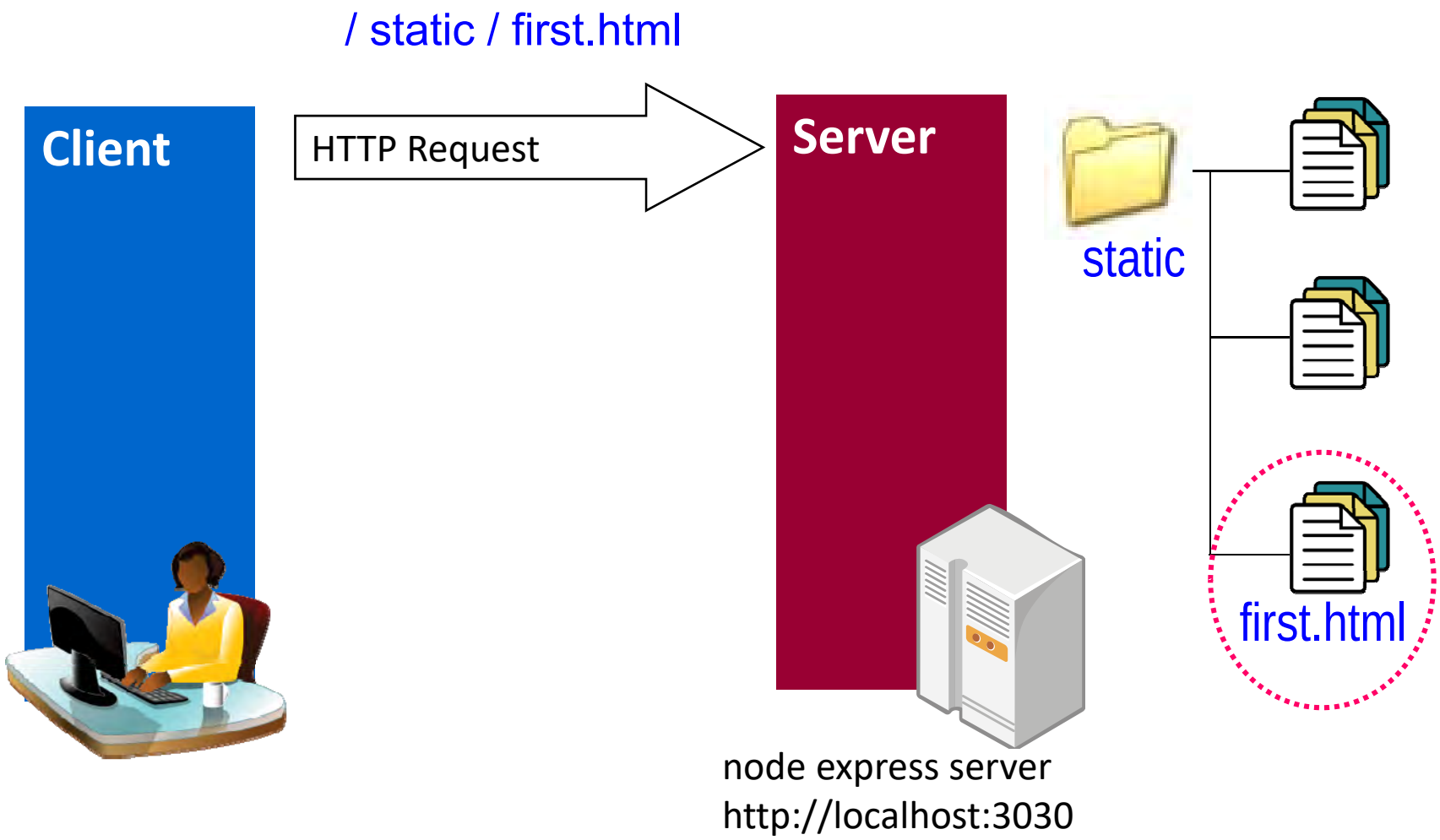
傳送靜態文件


/ static / first.html
Client
HTTP Request
Server
static
HTTP Response
first.html
node express server
http://localhost:3030

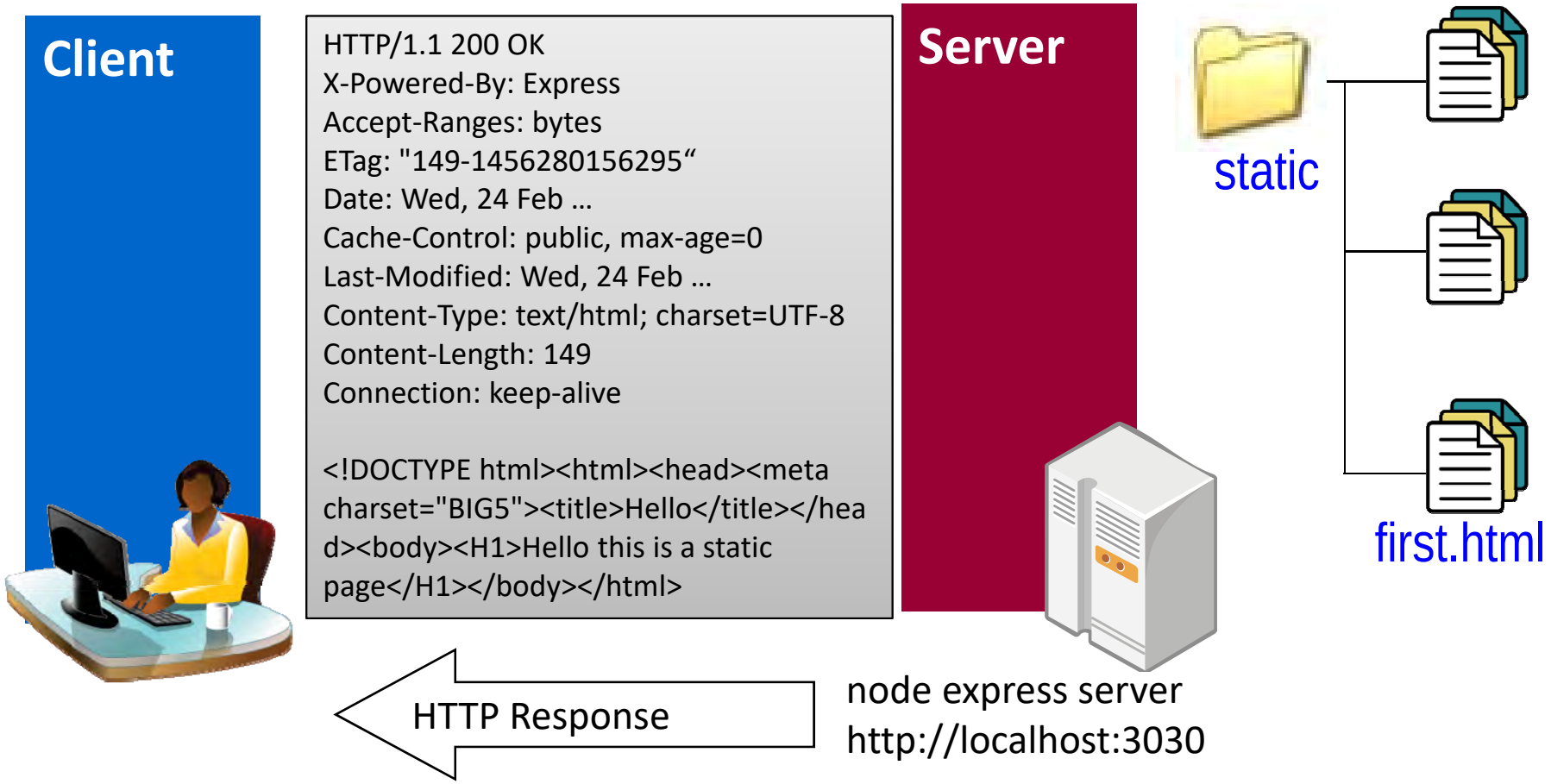
傳送靜態文件


/ static / first.html
Client
HTTP Request
GET /static/first.html HTTP/1.1
Host: localhost:3000
User-Agent: curl/7.44.0
Accept: */*
Server
static
first.html
node express server
http://localhost:3030

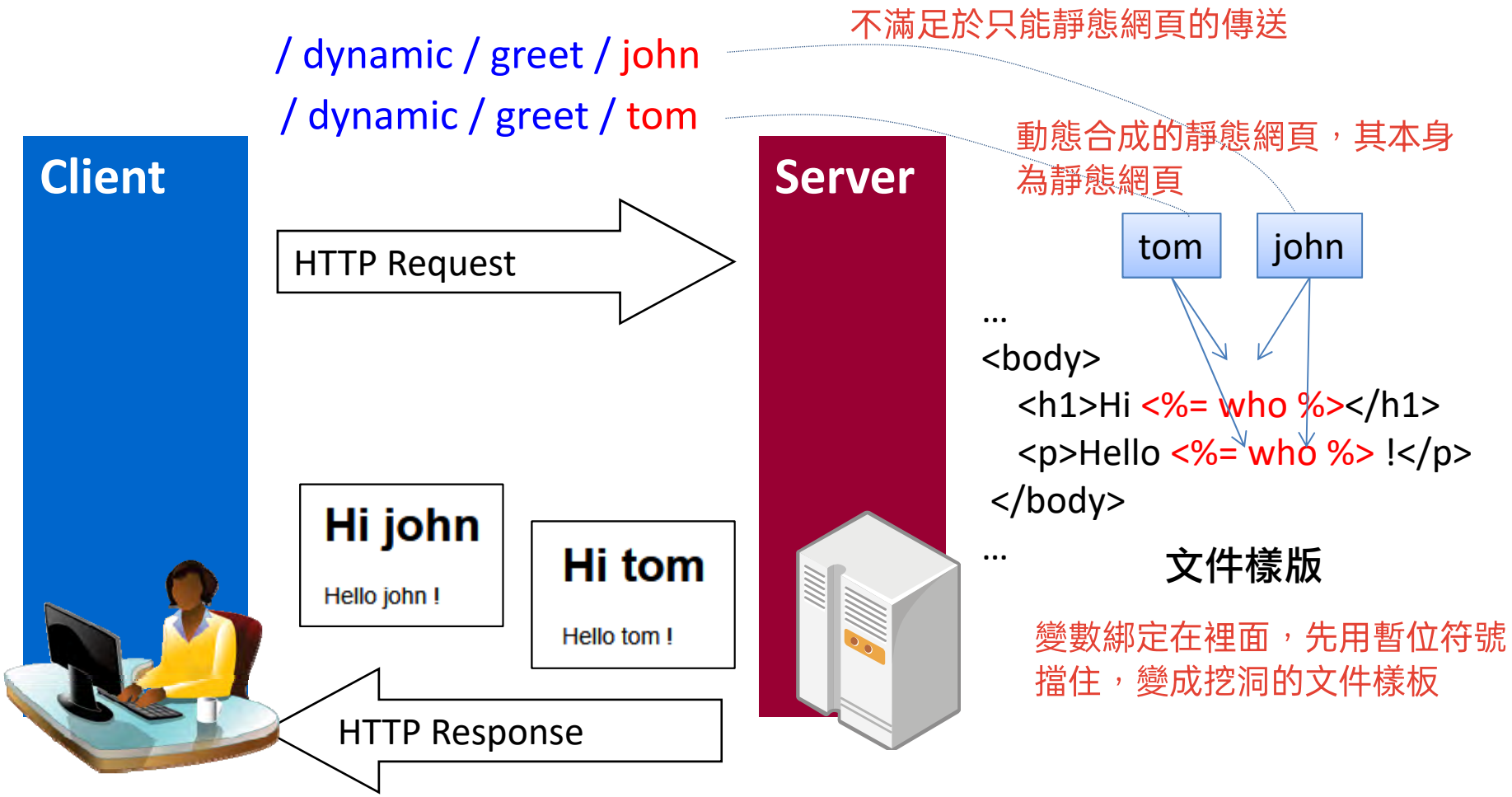
傳送靜態文件


/ static / first.html
Client
HTTP Request
Server
static
first.html
node express server
http://localhost:3030

傳送靜態文件


Client
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
ETag: "149-1456280156295“
Date: Wed, 24 Feb …
Cache-Control: public, max-age=0
Last-Modified: Wed, 24 Feb …
Content-Type: text/html; charset=UTF-8
Content-Length: 149
Connection: keep-alive
<!DOCTYPE html><html><head><meta charset="BIG5"><title>Hello</title></head><body><H1>Hello this is a static page</H1></body></html>
Server
static
first.html
HTTP Response
node express server
http://localhost:3030

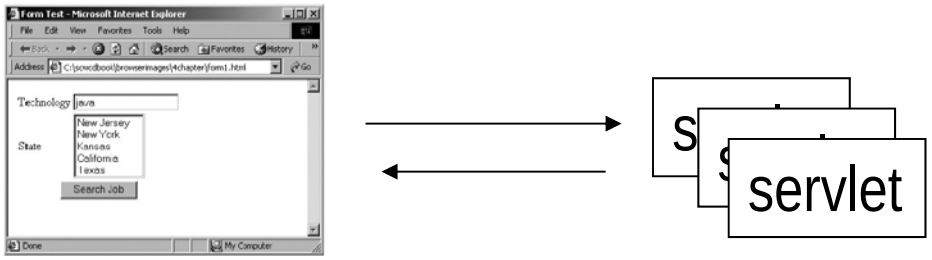
隨需合成文件: JSP/PHP



Servlets與JSP

邏輯和網頁寫在一起

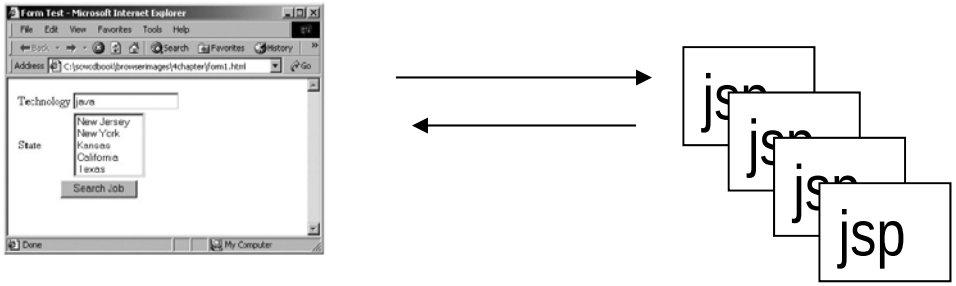
Pros: 不改變原來程式語法
Cons: 不知道印出來 UI 是如何(要對齊)




```
out.println(“<table><tr>”);
out.println(“<td>”+i+”</td>”);
out.println(“<td>”+j+”</td>”);
out.println(“</tr></table>”);
```

UI 很難進行改動

以程式碼為主，內嵌 HTML




```
<b>There are  
<% int i = getI();
     i++;
     out.println(i);
%> parking spaces left.</b>
```

也很亂，難以讀出在做什麼，比上面還糟

以 HTML 為主，內嵌程式碼

將Presentation Layer為主的畫面輸出嵌在程式碼中所造成的亂象

```
out.println("<body>");

printPullDownMenu();


out.println("<table width=\"100%\" height=\"100%\" border=\"0\" cellpadding=\"0\" cellspacing=\"0\"
  bgcolor=\"#FFFFFF\">");

out.println("<tr>");

out.println("<td height=\"59\" colspan=\"3\" background=\""+apserver+"SSO/images/title_bg.jpg\"
  bgcolor=\"#336699\" style=\"border-bottom:1px solid #000000;background-repeat:no-repeat;background-
  position:center left\">");

printHeader();

out.println("</td></tr><tr><td colspan=\"2\" valign=\"top\">");

printMessageBar(message);

out.println("</td></tr>");

out.println("<tr align=\"left\">");

out.println("<td width=\"73\"></td><td valign=\"top\" align=\"left\">");

  …
```

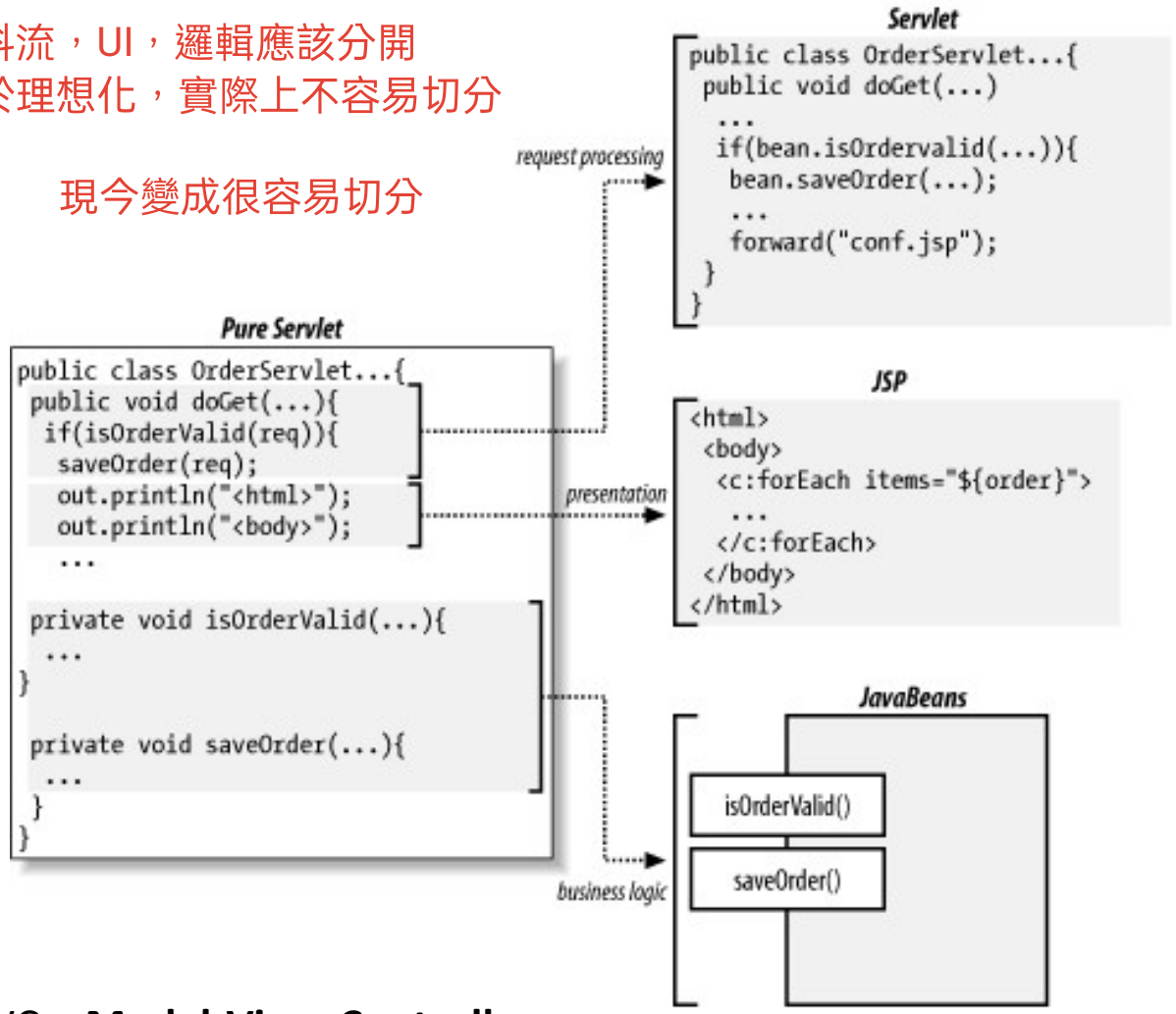
JSP Scriptlet造成程式碼難以維護

```
<%
    String name = null;
    if (request.getParameter("name") == null) {
%>
<%@ include file="error.html" %>
<%
    } else {
    foo.setName(request.getParameter("name"));
    if (foo.getName().equalsIgnoreCase("integra"))
    name = "acura";
    if (name.equalsIgnoreCase( "acura" )) {
%>
```

Separation of Concern

資料流，UI，邏輯應該分開
過於理想化，實際上不容易切分

現今變成很容易切分



C　解讀使用者的資料，
決定網頁流程

流程、業務邏輯
(業務邏輯較複雜時，另外放 M)

V　Html網頁

顯示邏輯

M　商務邏輯、資料

MVC = **Model-View-Controller**

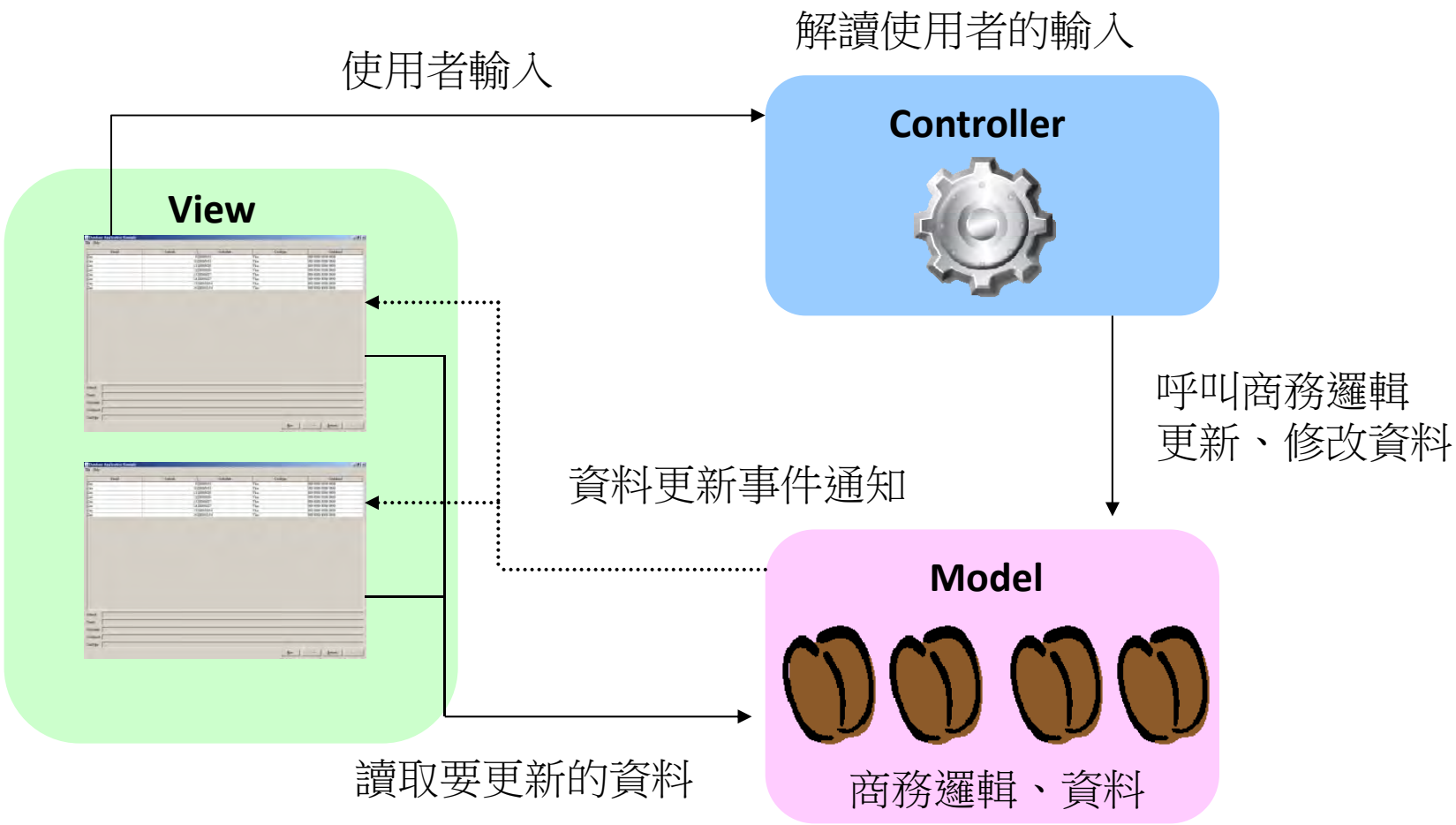
MVC架構

- T.Reenskaug在1979年發展出的GUI架構
- 早期Xerox PARC開發的視窗系統中，MVC被用來管理GUI

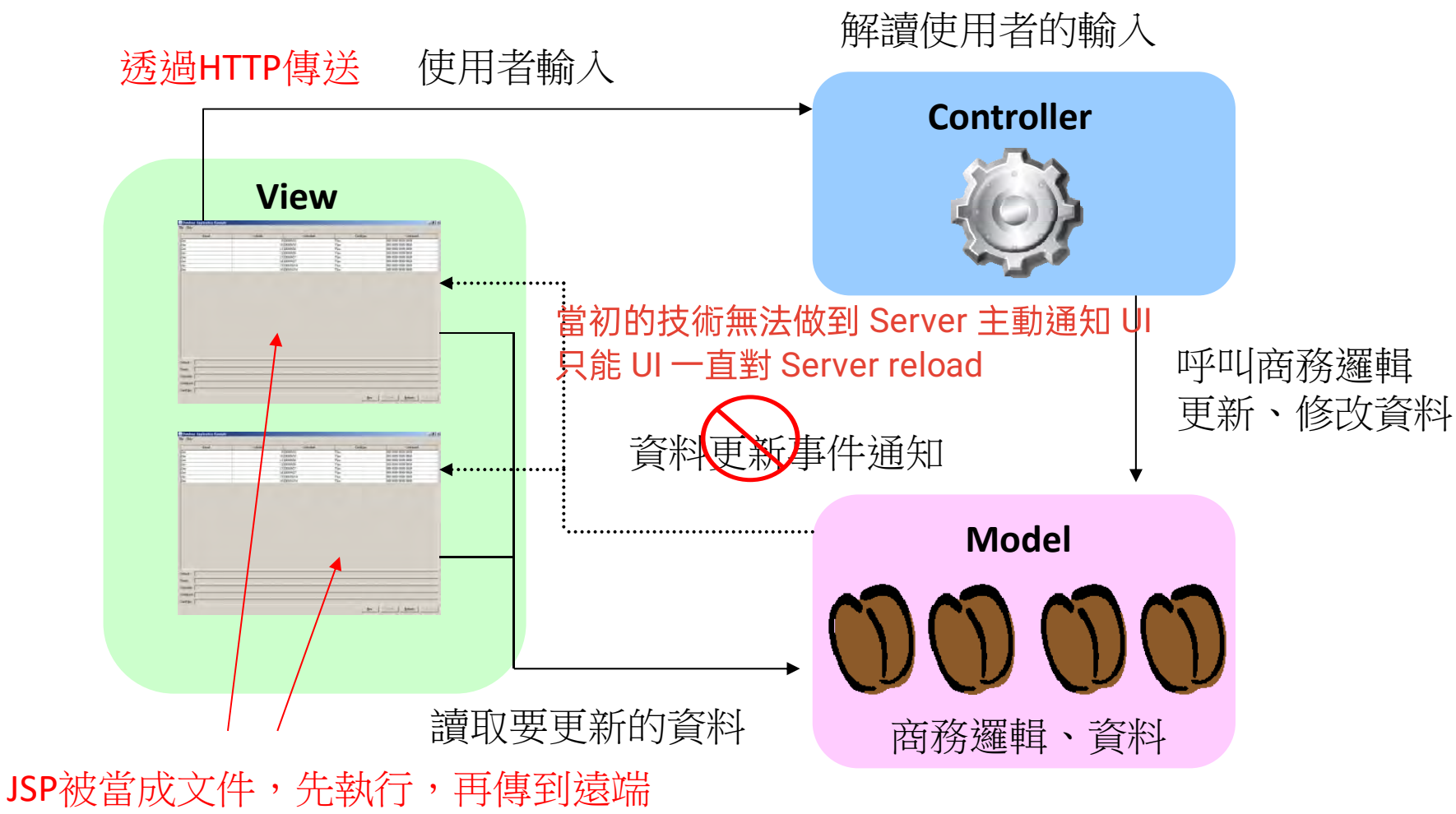
使用MVC的好處

- 將不同性質的邏輯清楚切分:
  - Business Logic (M)
  - Output Presentation (V)
  - Request Processing (C)
- 在單點控制流程
- 增進程式可維護性
- 增進程式的可擴充性

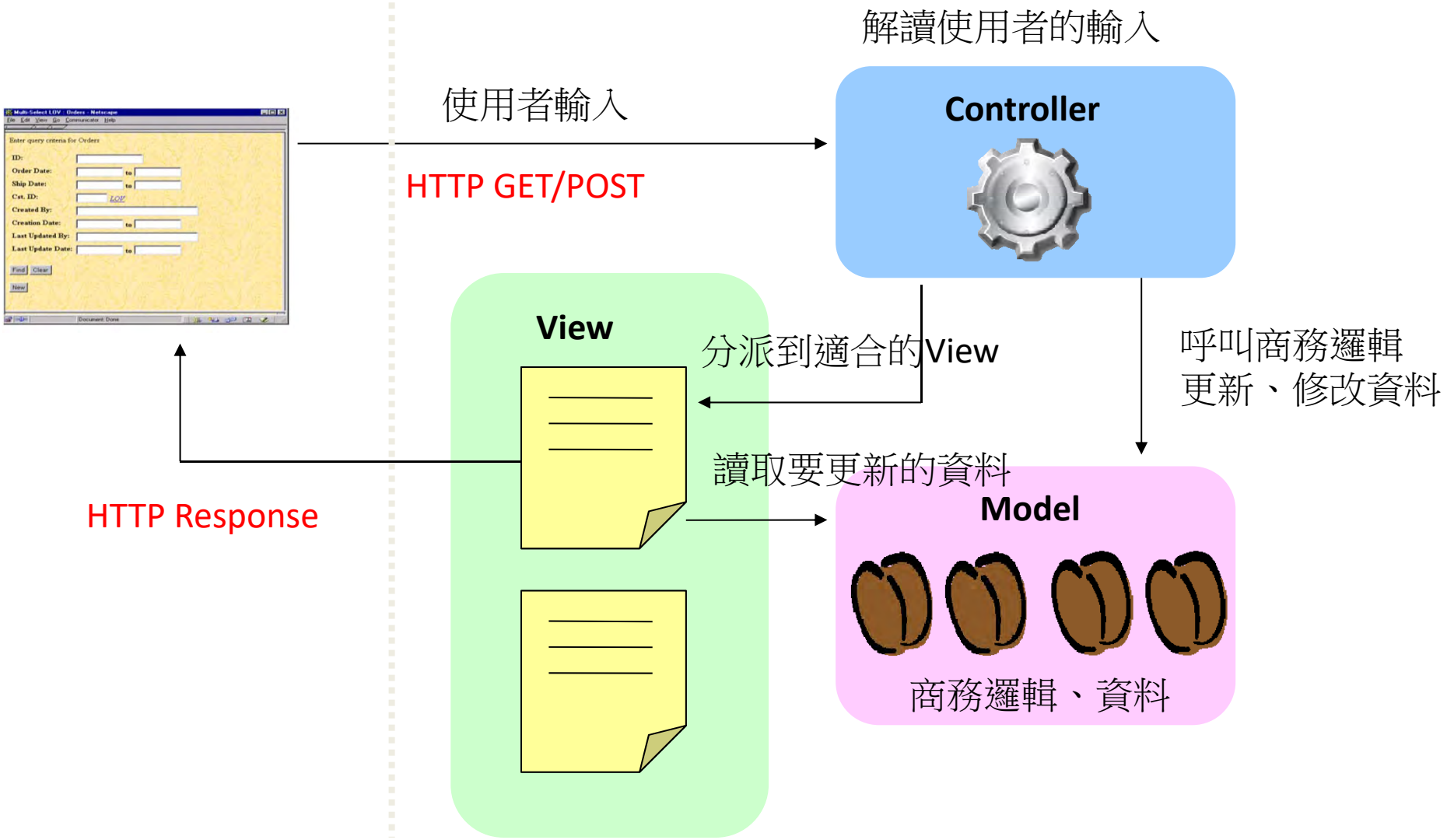
MVC (Desktop)


解讀使用者的輸入
使用者輸入
Controller
View
呼叫商務邏輯
更新、修改資料
資料更新事件通知
Model
讀取要更新的資料
商務邏輯、資料

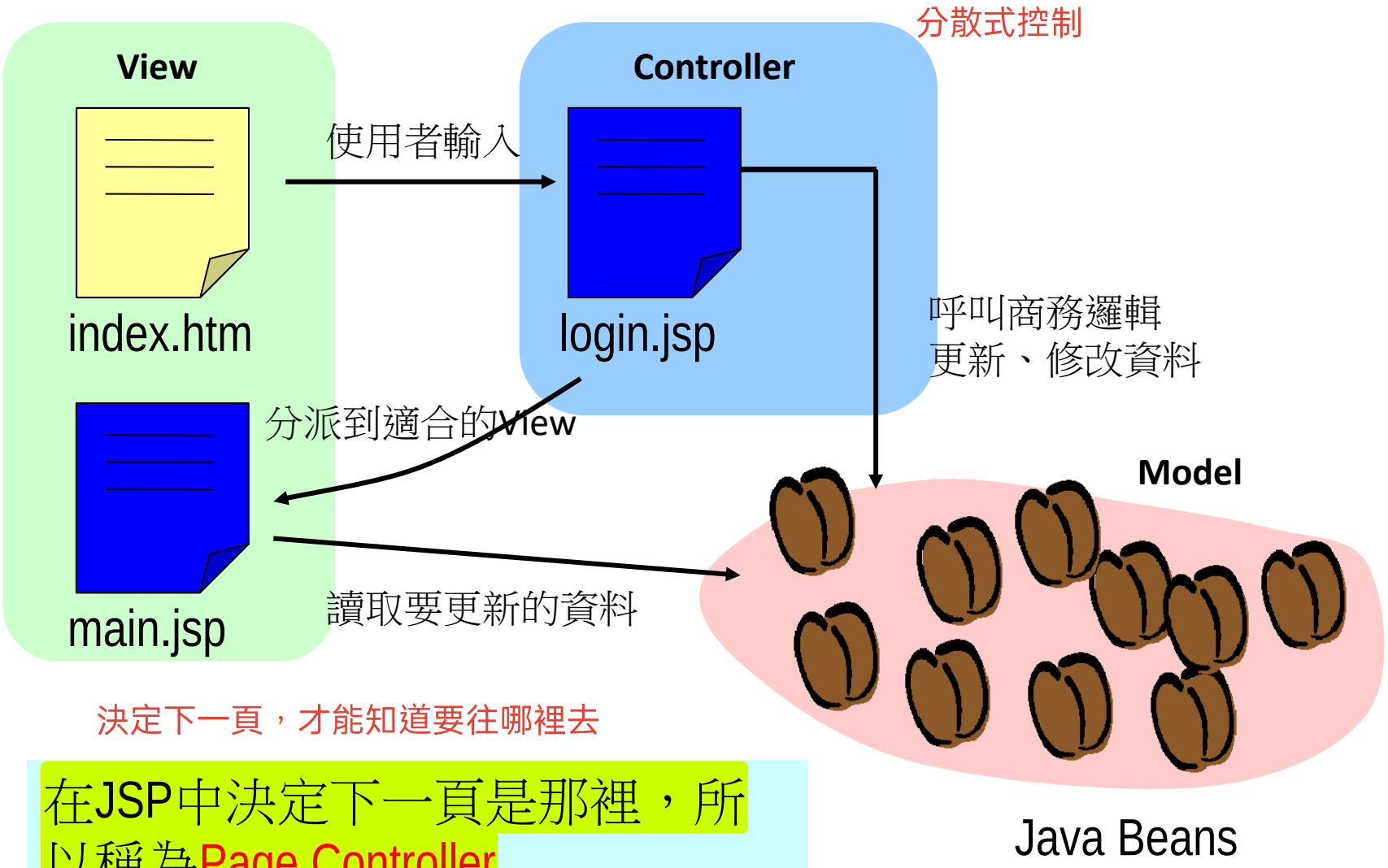
MVC應用在Web上問題



MVC (Web)


解讀使用者的輸入
使用者輸入
HTTP GET/POST
Controller
View
分派到適合的View
呼叫商務邏輯
更新、修改資料
讀取要更新的資料
HTTP Response
Model
商務邏輯、資料

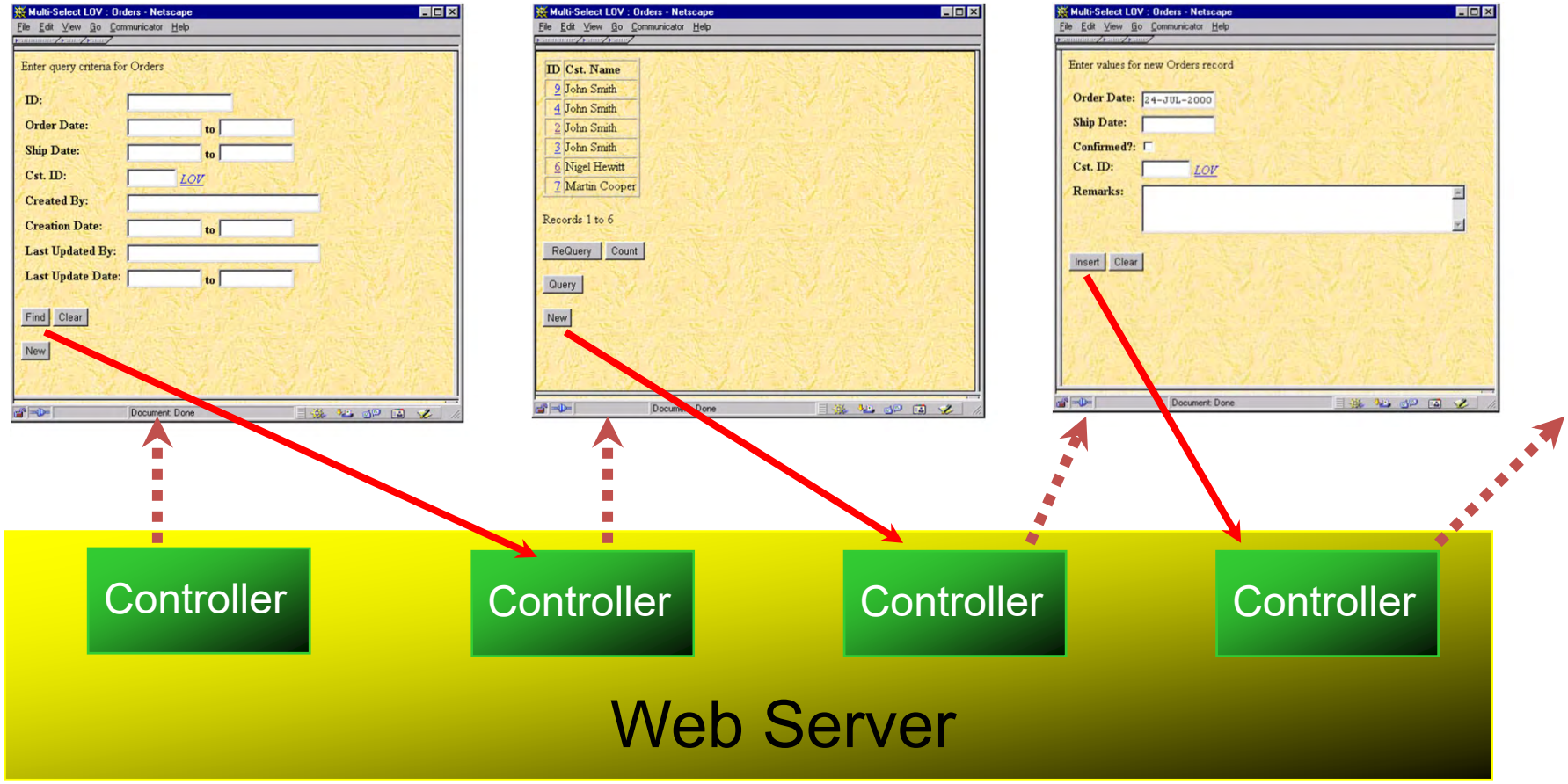
Model 1 : Page Controller



在JSP中決定下一頁是那裡，所以稱為Page Controller

Model 1有什麼問題？

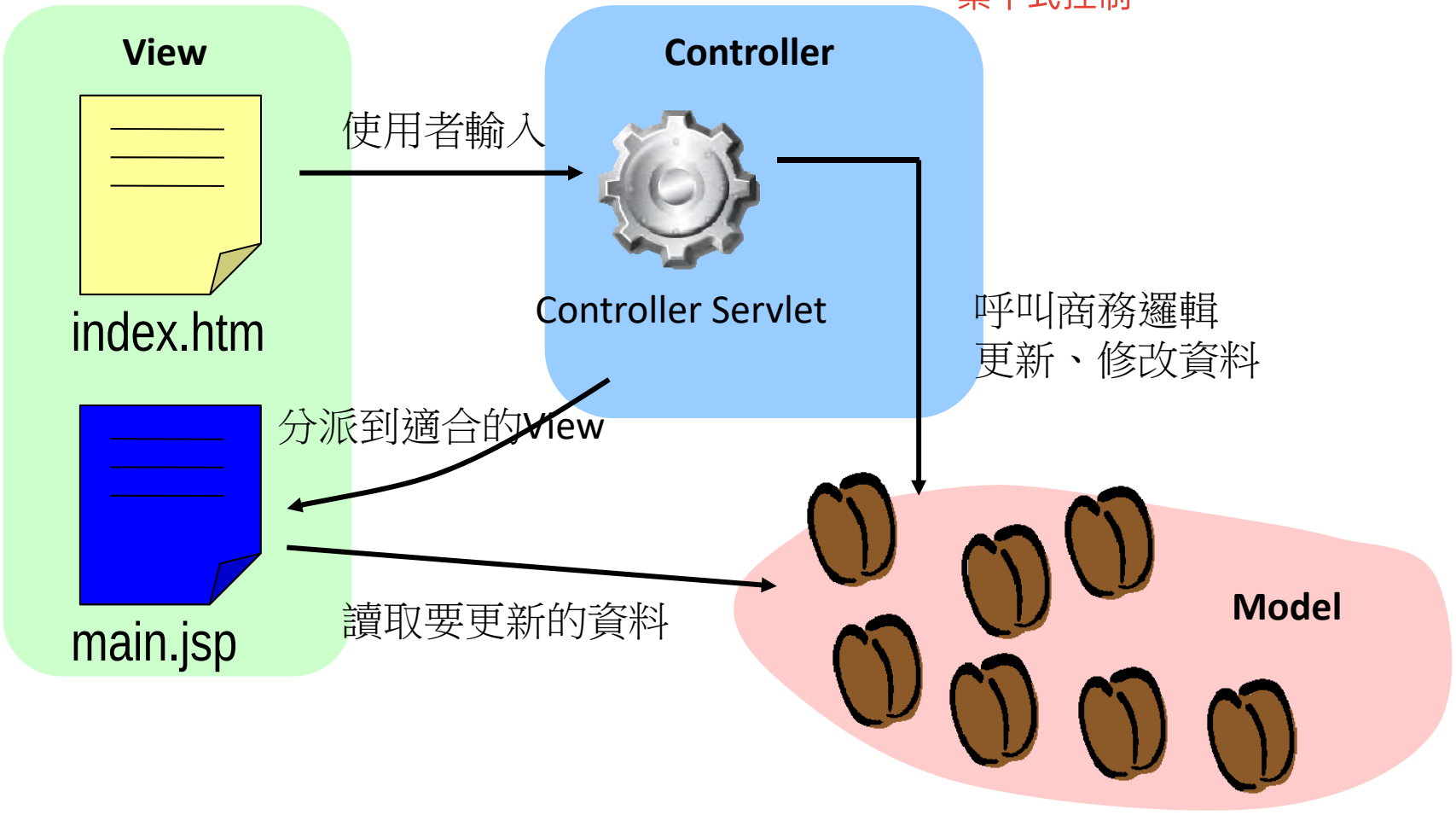
One controller per page , hard to maintain !



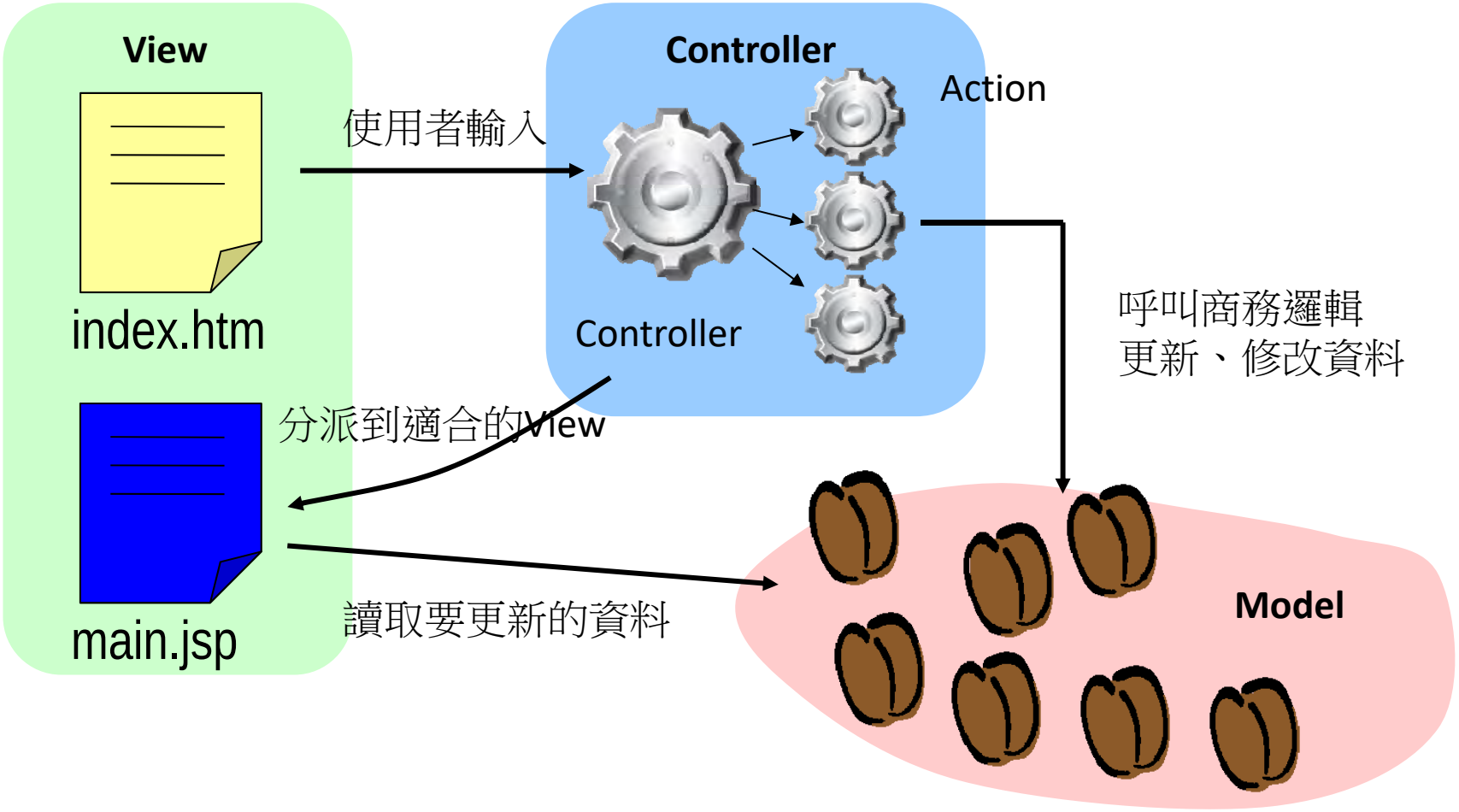
現今常用此

Model 2

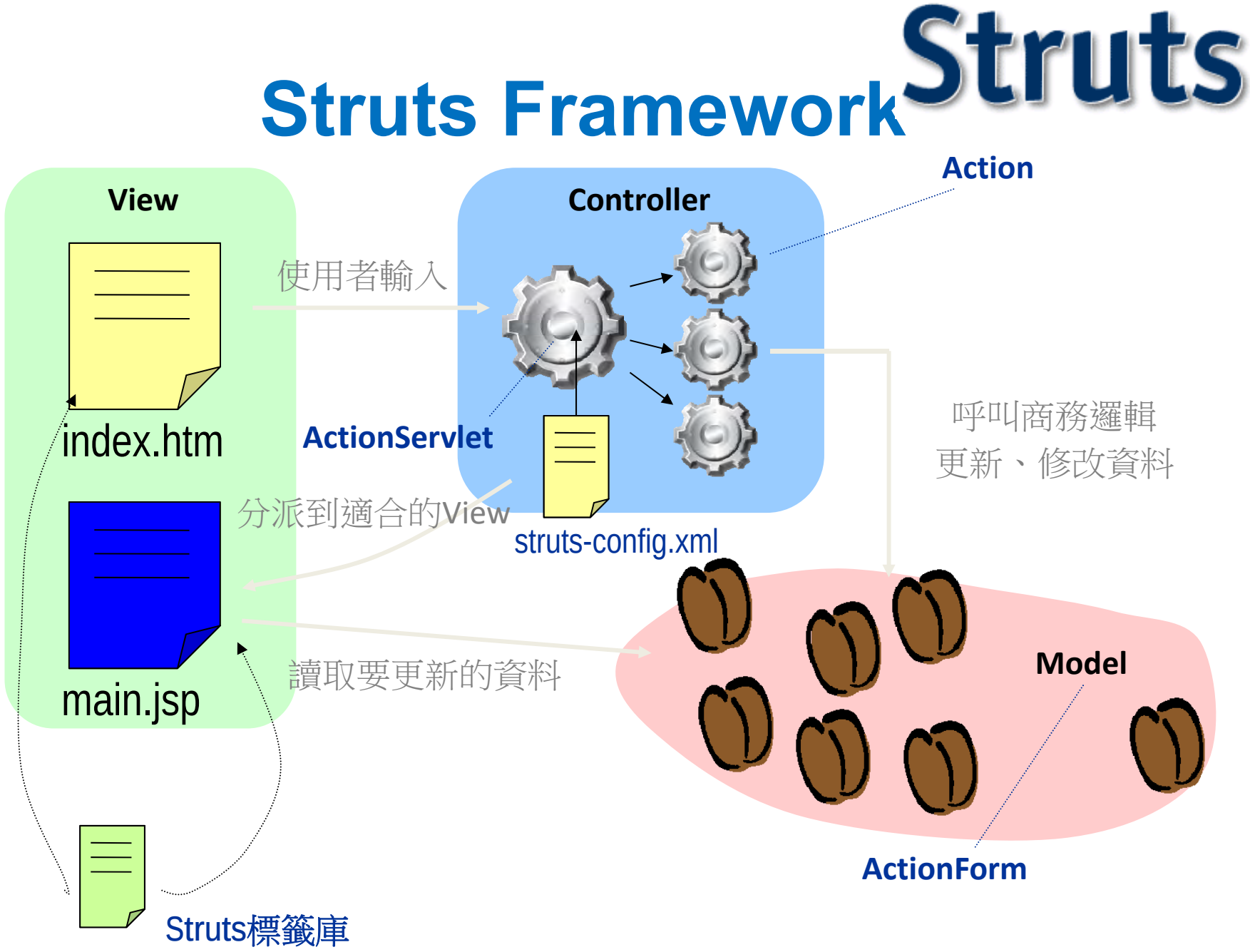
集中式控制



將Controller模組化

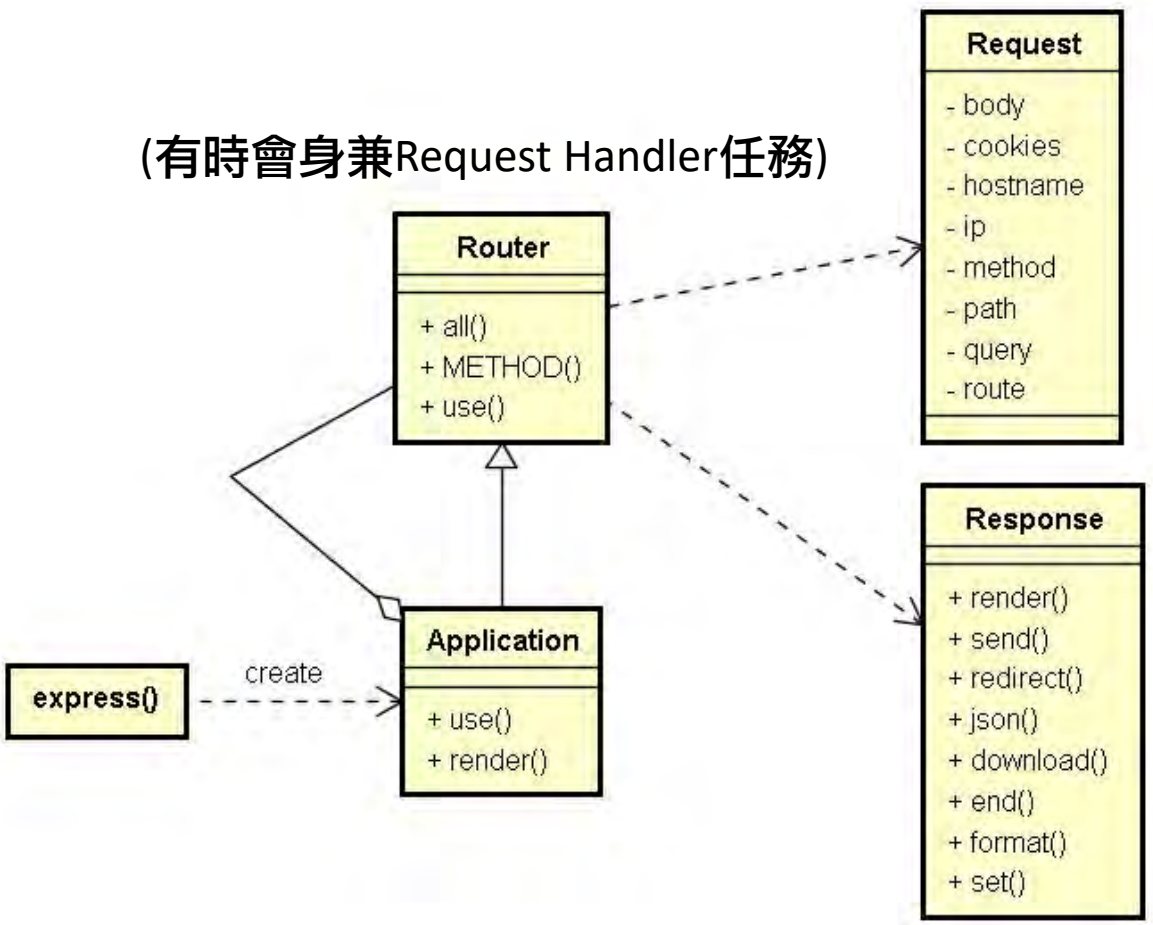


Struts Framework

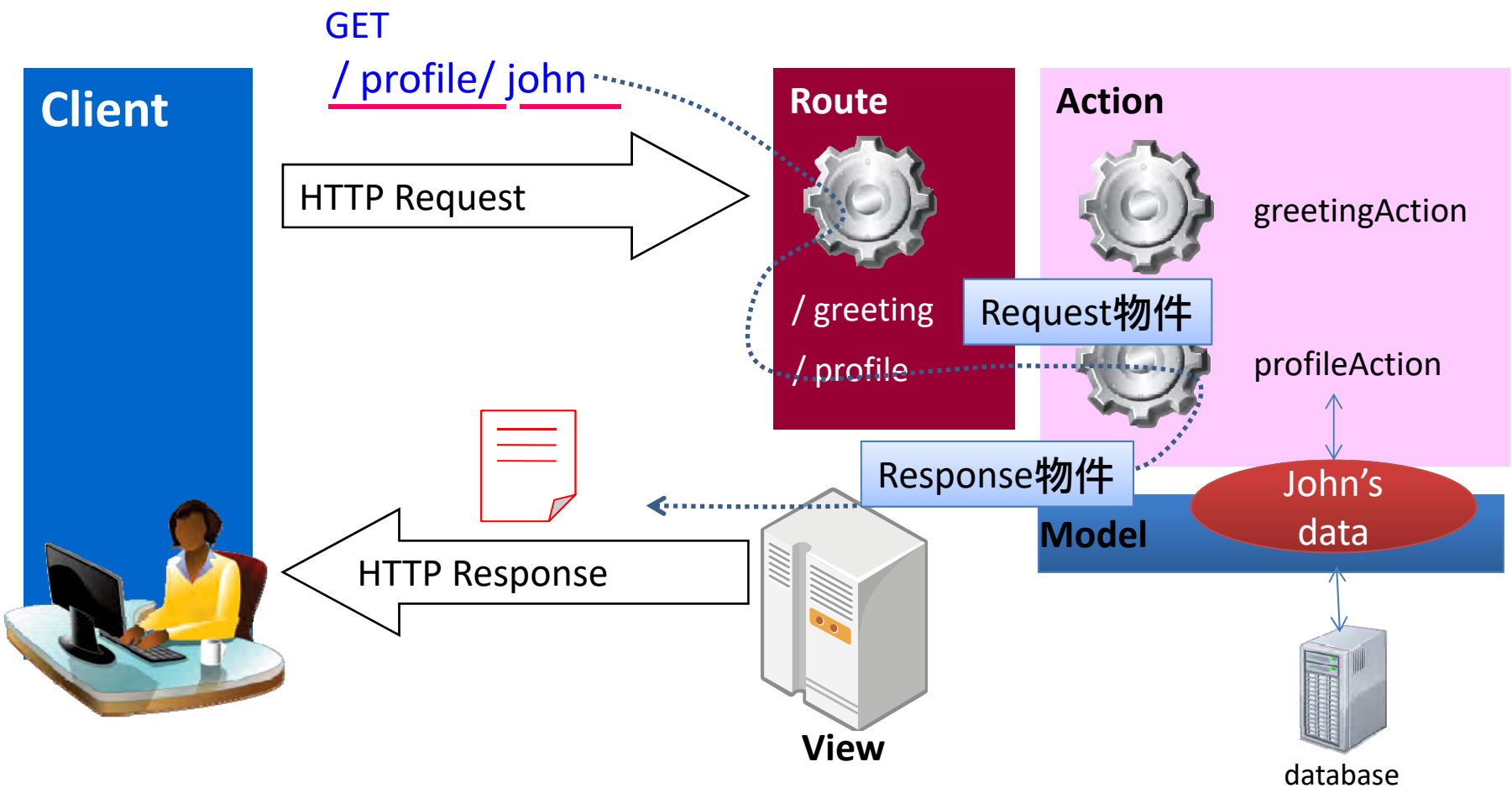


Express結構

(有時會身兼Request Handler任務)

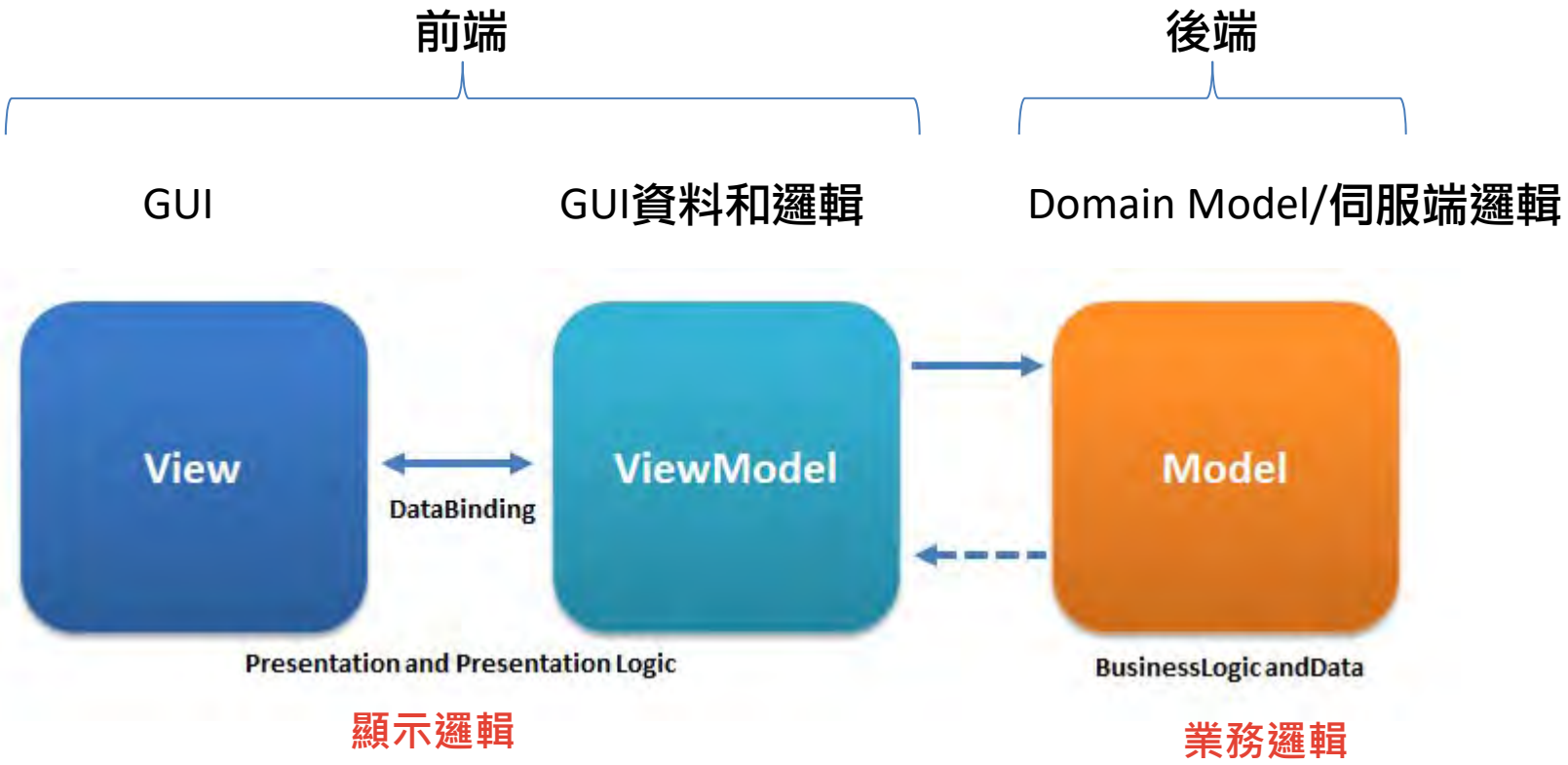


Express MVC處理架構



MVVM

Introducing ViewModel and Binder
ViewModel 和 Model間料不一定要即時同步，因此交通量會較MVC低
將大部份GUI 事務由後台切割出來 (後台不再管理Controller)



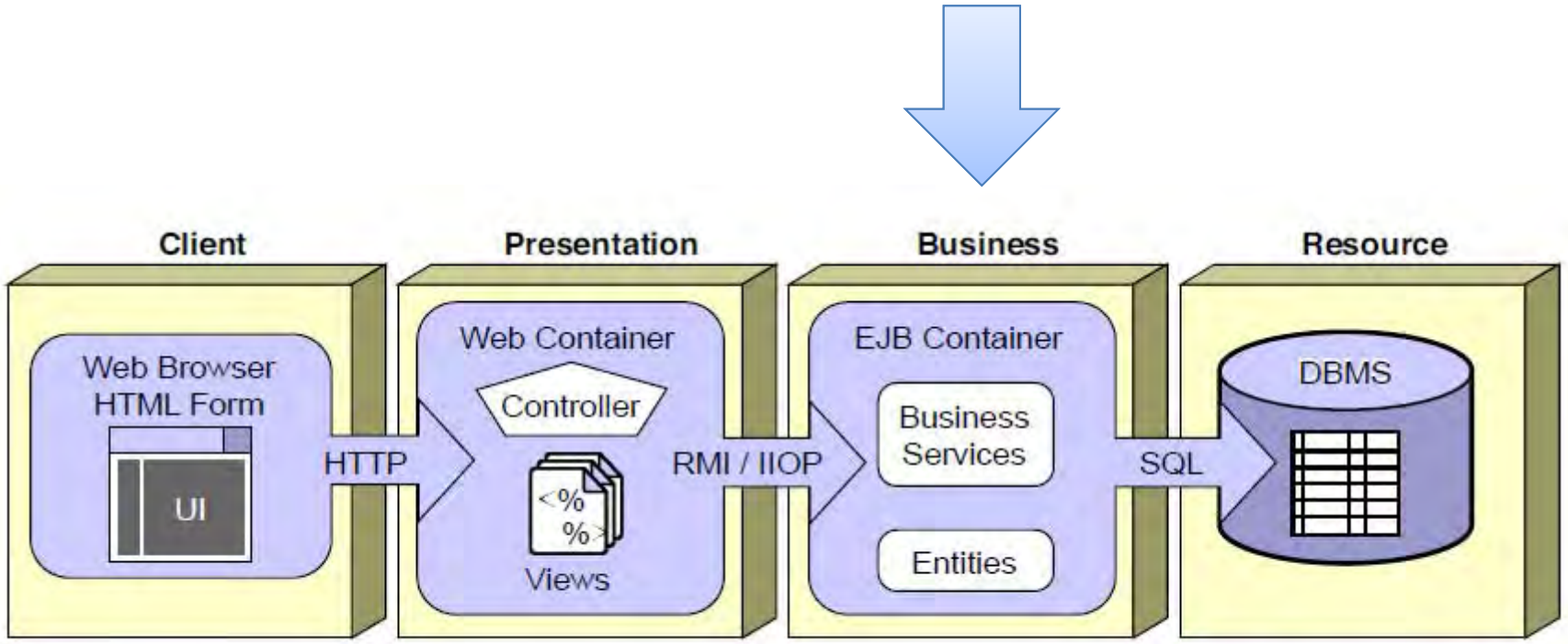
Summary

- 傳統Web應用程式架構 (MVC)
  - Web client/server的互動主要是client向server下達GET抓取網頁或透過POST貼回資料
  - MVC大都在後台
    - Controller管控應用程式流程，分配View
    - View 少部份邏輯可能在前台
- 當代Web應用程式架構 (MVVM)
  - Web client/server的互動主要是client存取server上的Web API
  - 只有Model在後台
    - 做為Web API提供者

後端的一些東西放進前端
(因為 Browser 的功能增強了，讓後端少負擔一些，這樣能連接更多設備)

中介服務架構

Enterpise JavaBeans
實作業務邏輯的元件



EJB Component Types

- Business logic
  - 可以想成remote function call
  - Session Beans (Stateless, stateful)
- Messaging
  - 可以想成remote event processor
  - Message Driven Beans
- Persistent entity
  - Entity Beans

```
int result = bean.add(1+1);
```

無狀態(client自帶狀態)

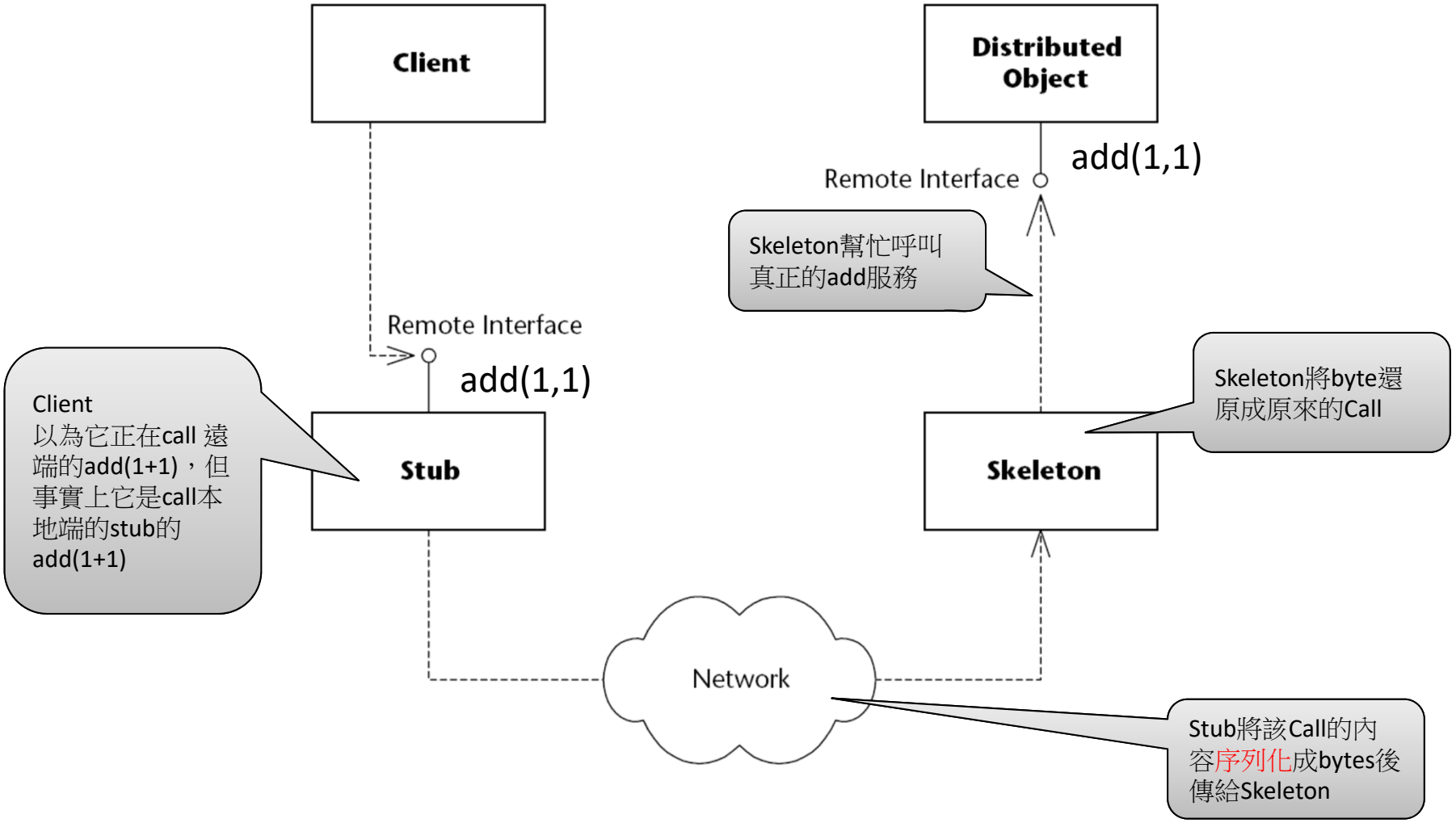
有狀態(server需另外存)

```
public  void
onMessage(Message message)
{
   ....

}
```

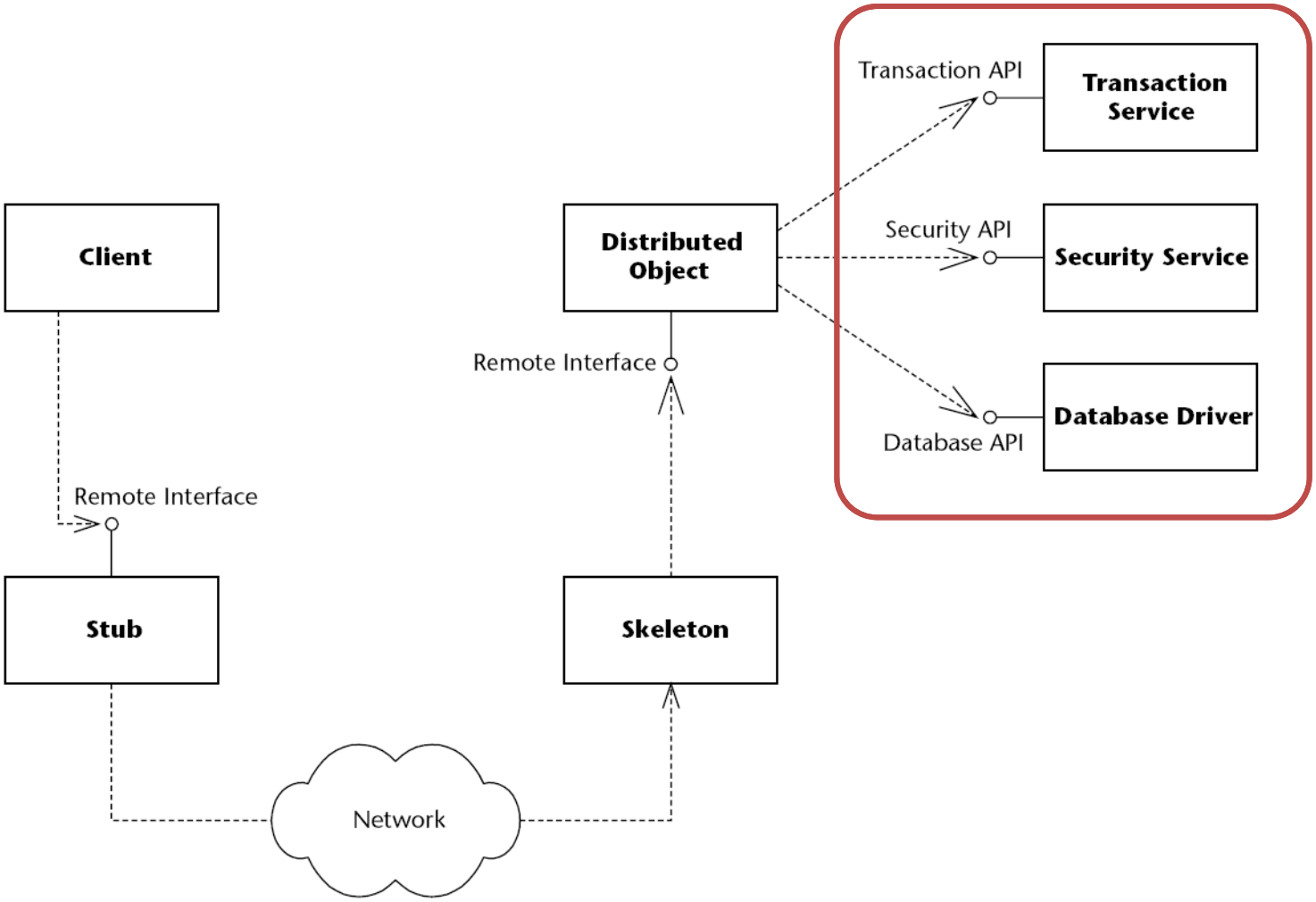
EJB Container做了些什麼?

- Accessing Remote Object
  - Stub/Skeleton架構
  - Serialization 物件序列化/反序列化
- 交易(Transaction)處理
- 安全檢查
- 記憶體/CPU 效能最佳化
  - Pooling and Activation/Passivation

典型呼叫遠端物件的架構

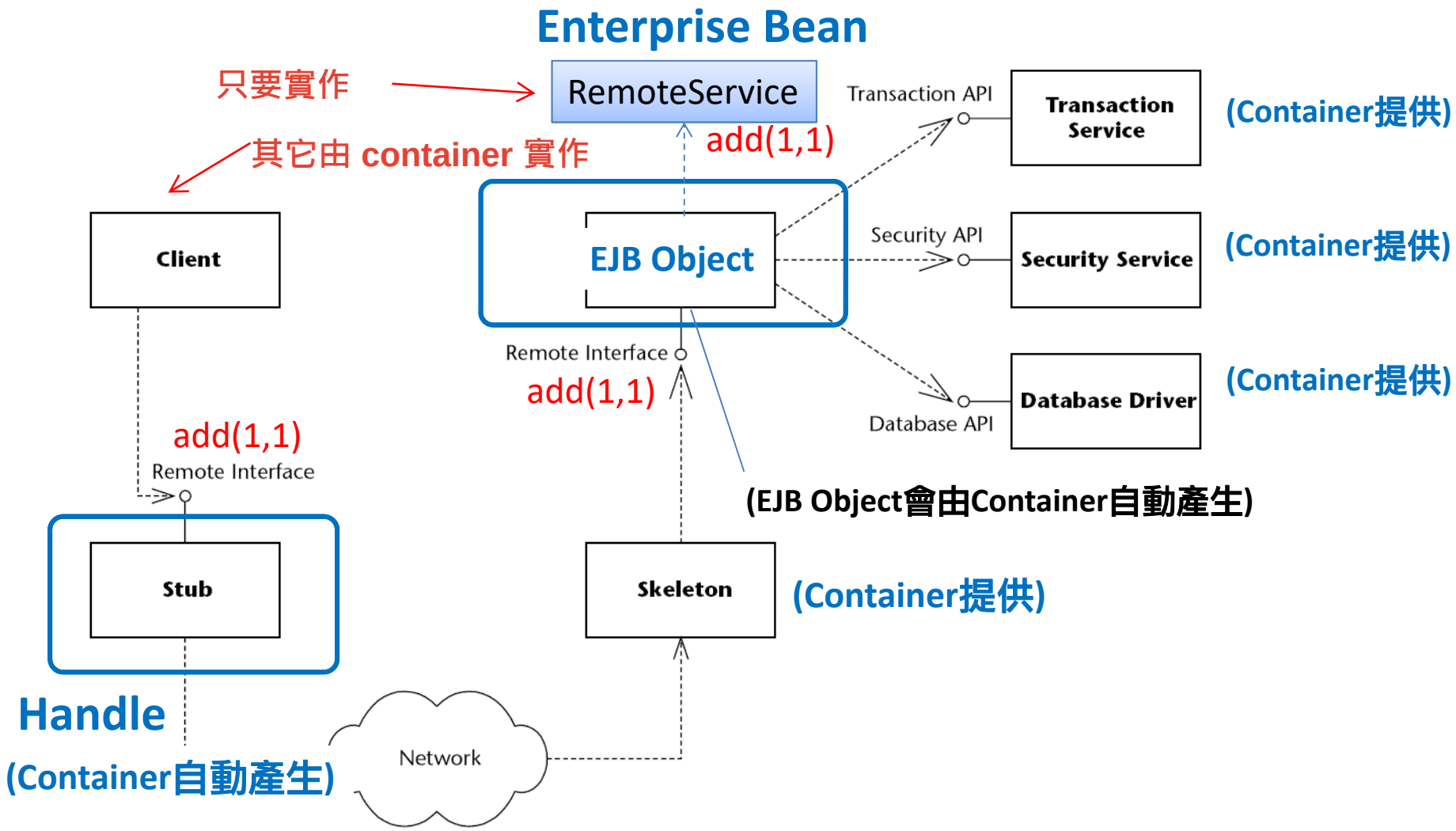


EJB容器在Skeleton中
順便幫我們加入了其它服務…

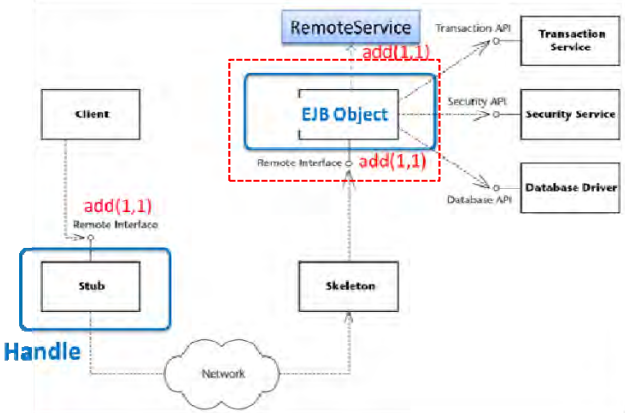


在EJB容器中我們這樣稱呼它們

寫程式容易，但配置困難

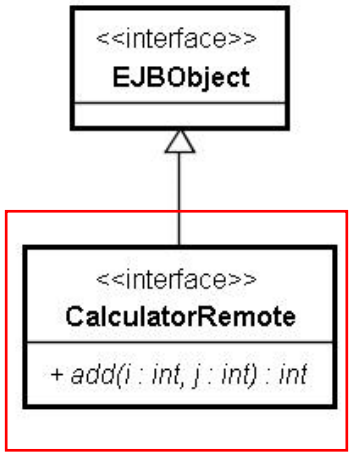


產生EJB Object



首先，你要宣告遠端服務的介面與方法，例如 public int add(int i, int j)
這個介面稱為Remote Interface

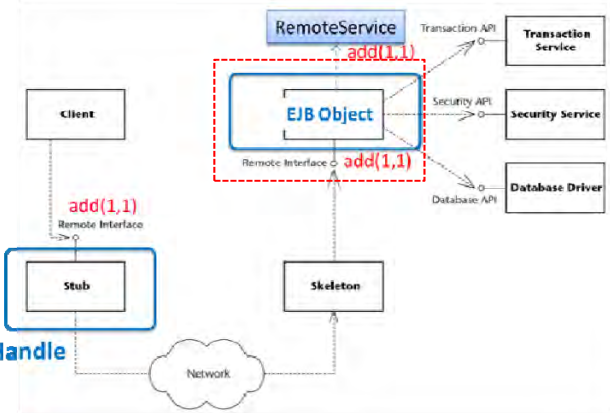
因為要讓Container自動產生EJB Object，所以你要標記它是一個EJB Object




```
public interface CalculatorRemote extends EJBObject {

    public int add(int i,int j) throws RemoteException;

}
```

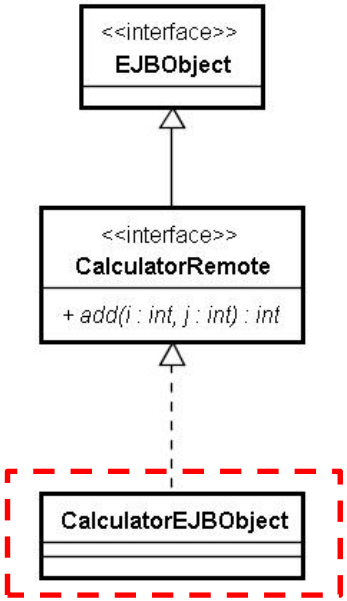
強迫開發者了解其為遠端連線
可能會斷線，所以要另外做 Error 處理

產生EJB Object



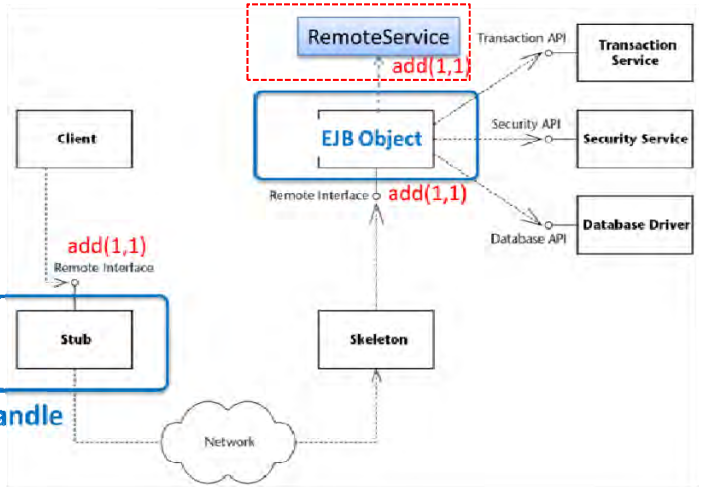
依照你宣告的interface內容，

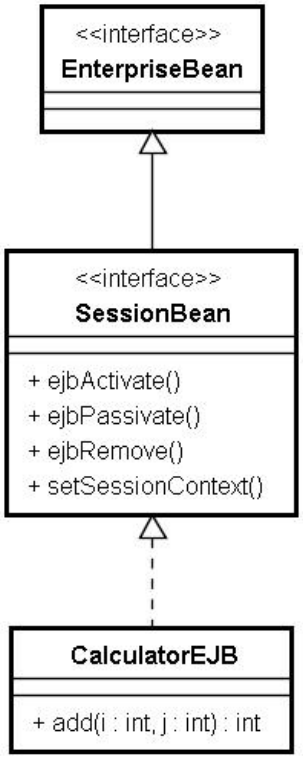
EJB容器會自動幫你產生專屬的EJBObject



自動產生

寫真正的服務

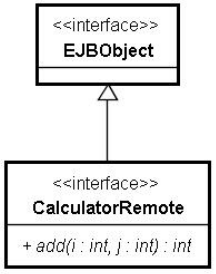




真正的服務必須要實作任意一個EnterpriseBean的子介面

例如"SessionBean"

參數要對得起來




```
public interface CalculatorRemote extends EJBObject {

   public int add(int i,int j) throws RemoteException;

}
```

另外，還要「手動」加上Remote Interface中所宣告methods的真正實作，也就是:

```
public int add(int i, int j) {
    return i + j;
}
```

EJB Home

- 平常要使用一個元件是直接new

```
Calculator calculator = new Calculator();
int result = calculator.add(1,1);
```

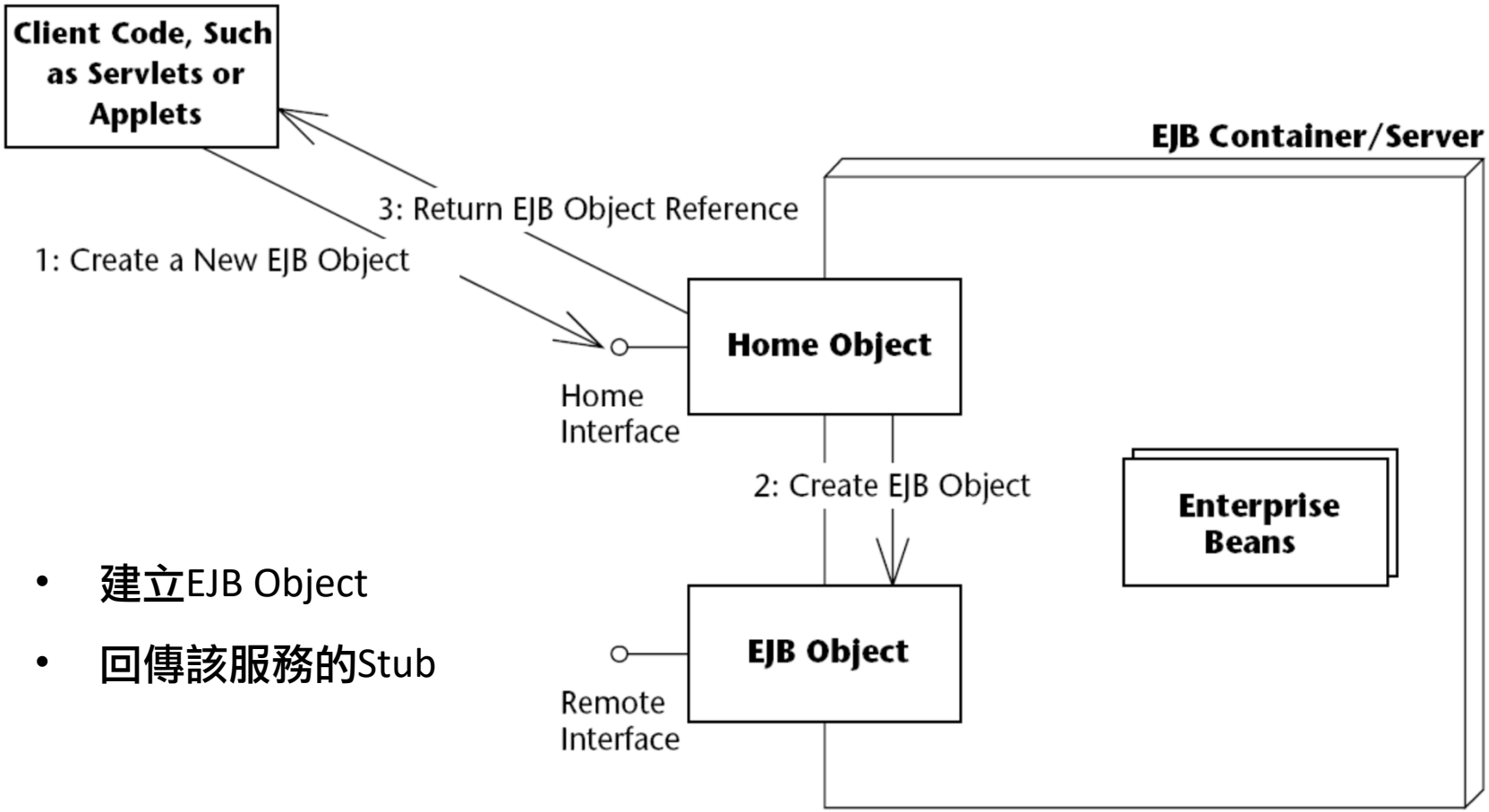
- 如何new遠端的物件?
  - 需要使用EJB Home

```
CalculatorHome calculatorHome = …;
Calculator calculator = calculatorHome.create();
int result = calculator.add(1,14);
```

EJB Home

EJB Home何處尋?

EJB Home的主要功能

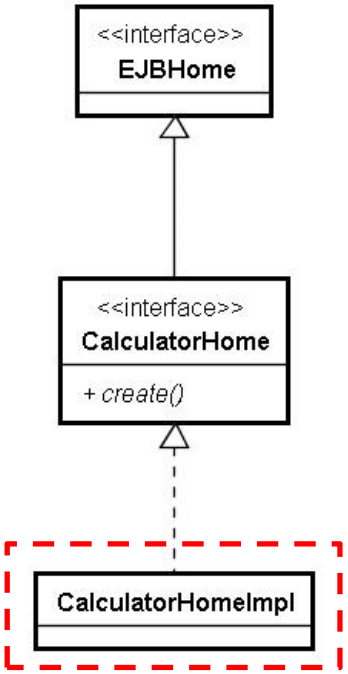


- 建立EJB Object
- 回傳該服務的Stub

寫EJB Home

依照你宣告的home interface內容，

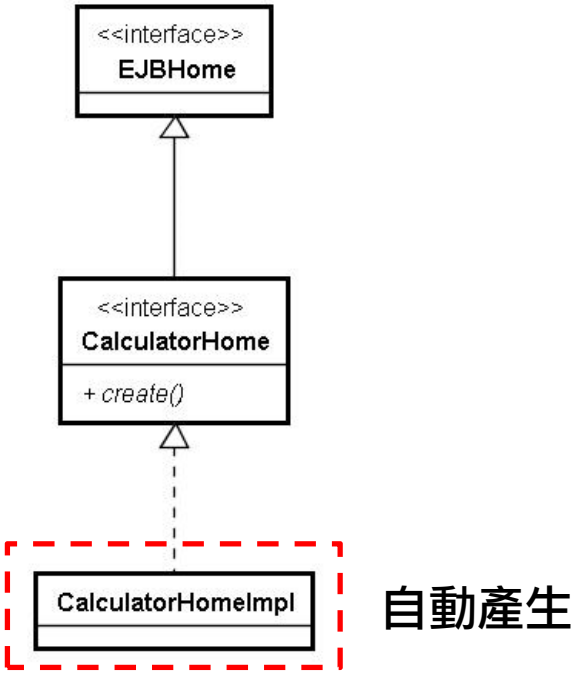
EJB容器會自動幫你產生專屬的EJB Home Implementation



自動產生

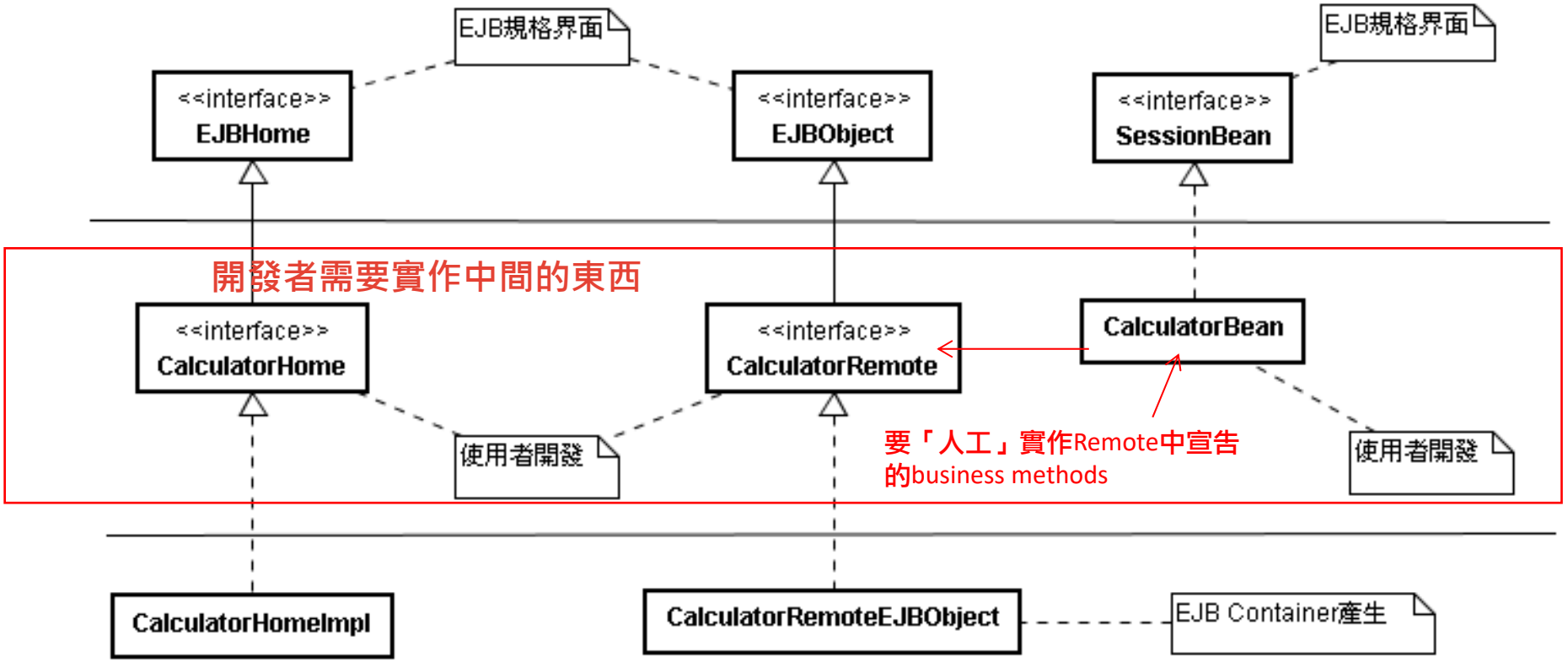
CalculatorHome

```
public interface CalculatorHome extends EJBHome {

    public CalculatorRemote create()
            throws RemoteException, CreateException;

}
```



自動產生

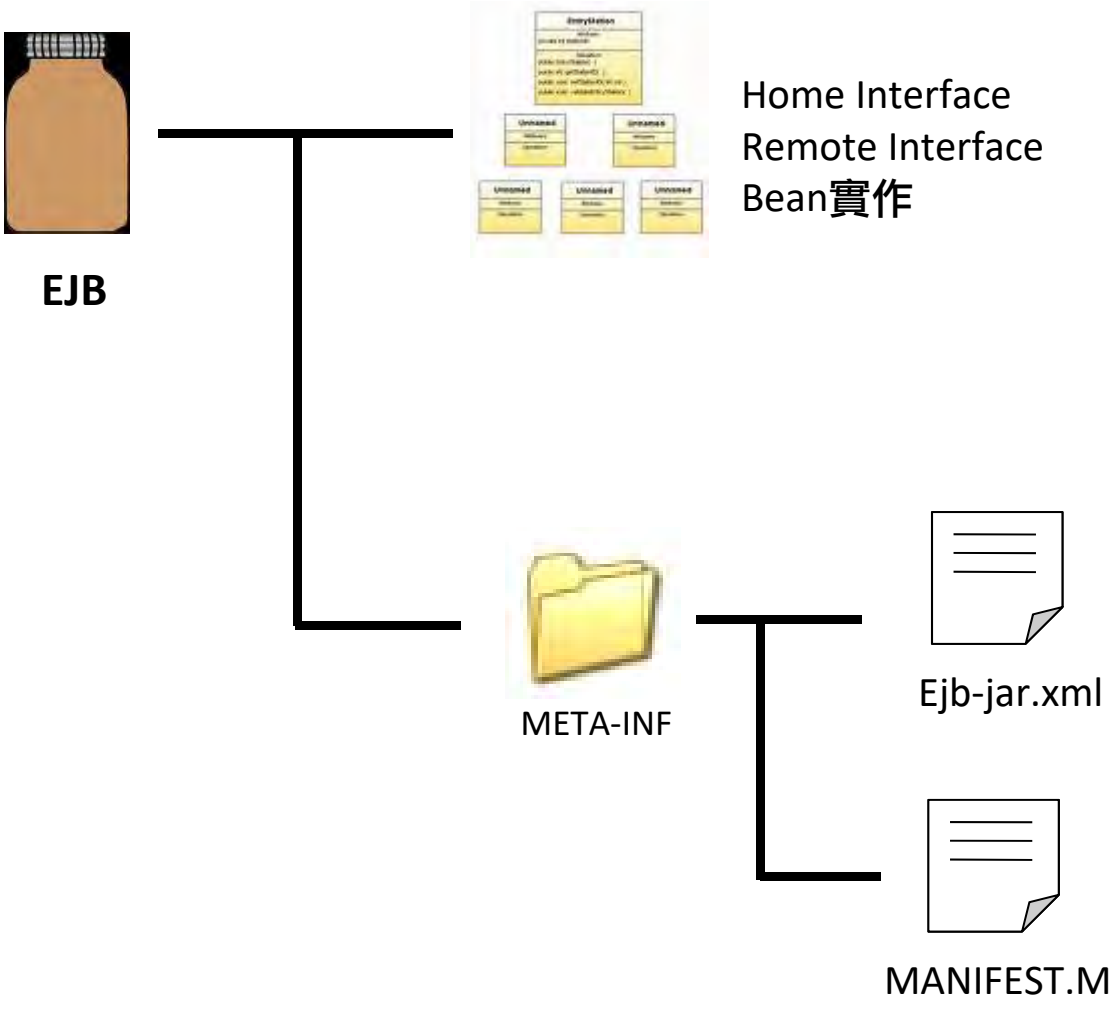
Summary
開發人員要宣告2個介面、實作1個類別



如何手工打造EJB元件

1. 開發:
   - 一個Home介面
   - 一個Remote或Local介面
   - 一個Enterprise Bean類別
2. 設定
   - 寫作佈署描述檔(ejb-jar.xml)
3. 打包
   - 將所有東西zip起來
4. 佈署
   - 放到Server上

EJB 元件實體結構


EJB
Home Interface
Remote Interface
Bean實作
META-INF
Ejb-jar.xml
MANIFEST.MF

Client

1. 設定JNDI naming context

```
Properties properties = new Properties();
properties.put(InitialContext.INITIAL_CONTEXT_FACTORY,
                    "com.sun.enterprise.naming.impl.SerialInitContextFactory");
InitialContext ic = new InitialContext(properties);
```

2. 從JNDI中取得EJB Home

```
Object ref = ic.lookup("fcu.scd.ejb20.CalculatorHome");
CalculatorHome calculatorHome = (CalculatorHome) PortableRemoteObject
                                                    .narrow(ref, CalculatorHome.class);
```

3. 用EJB Home建立遠端元件實體，取得Stub，並加以呼叫

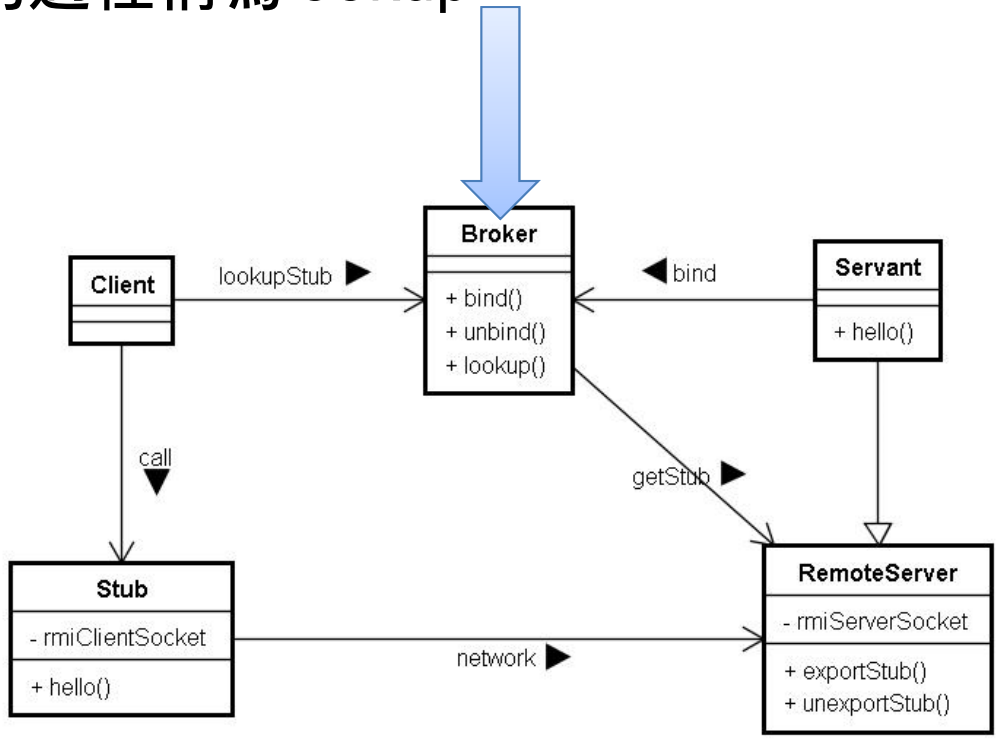
```
CalculatorRemote calculator = calculatorHome.create();
int result = calculator.add(2, 4);
```

Java Naming and Directory Interface (JNDI)

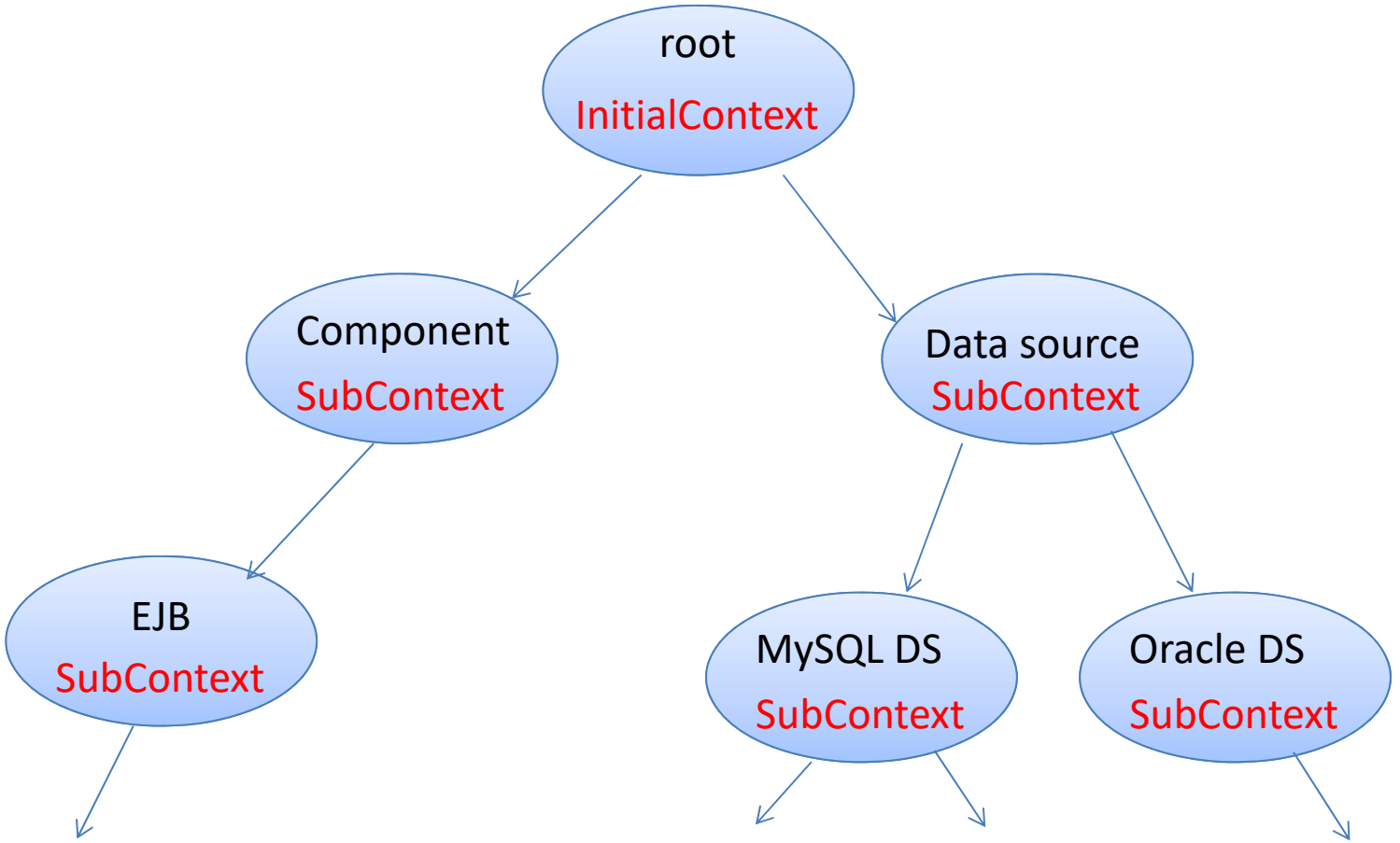
- Finding any object with a “name”
  - 電腦系統辨識物件的方法
    - Local: reference
    - Remote: remote reference (IP address, port)
  - JNDI將所有物件貼上「名稱(name)」標籤
    - 可透過「名稱」取得物件
    - 較易於人類所理解、記憶
    - 便於伺服器管理人員進行維護工作

Java Naming and Directory Interface (JNDI)

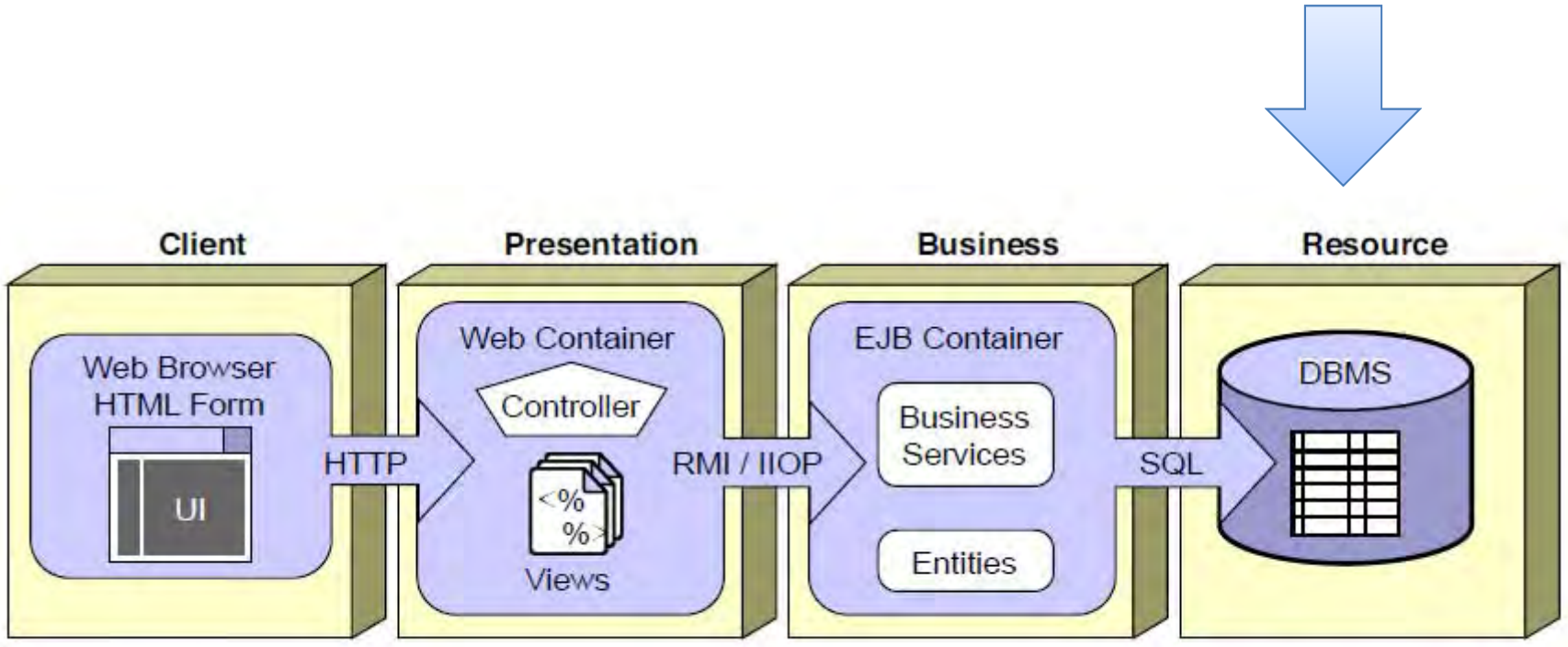
- Naming Service
  - 很像查號台，告訴他name，他告訴你電話號碼
  - 此一查詢過程稱為lookup



JNDI Tree


root
InitialContext
Component
SubContext
Data source
SubContext
EJB
SubContext
MySQL DS
SubContext
Oracle DS
SubContext

資料存取架構


Client
Presentation
Business
Resource
Web Browser
HTML Form
UI
HTTP
Web Container
Controller
<%
%>
Views
RMI / IIOP
EJB Container
Business
Services
Entities
SQL
DBMS

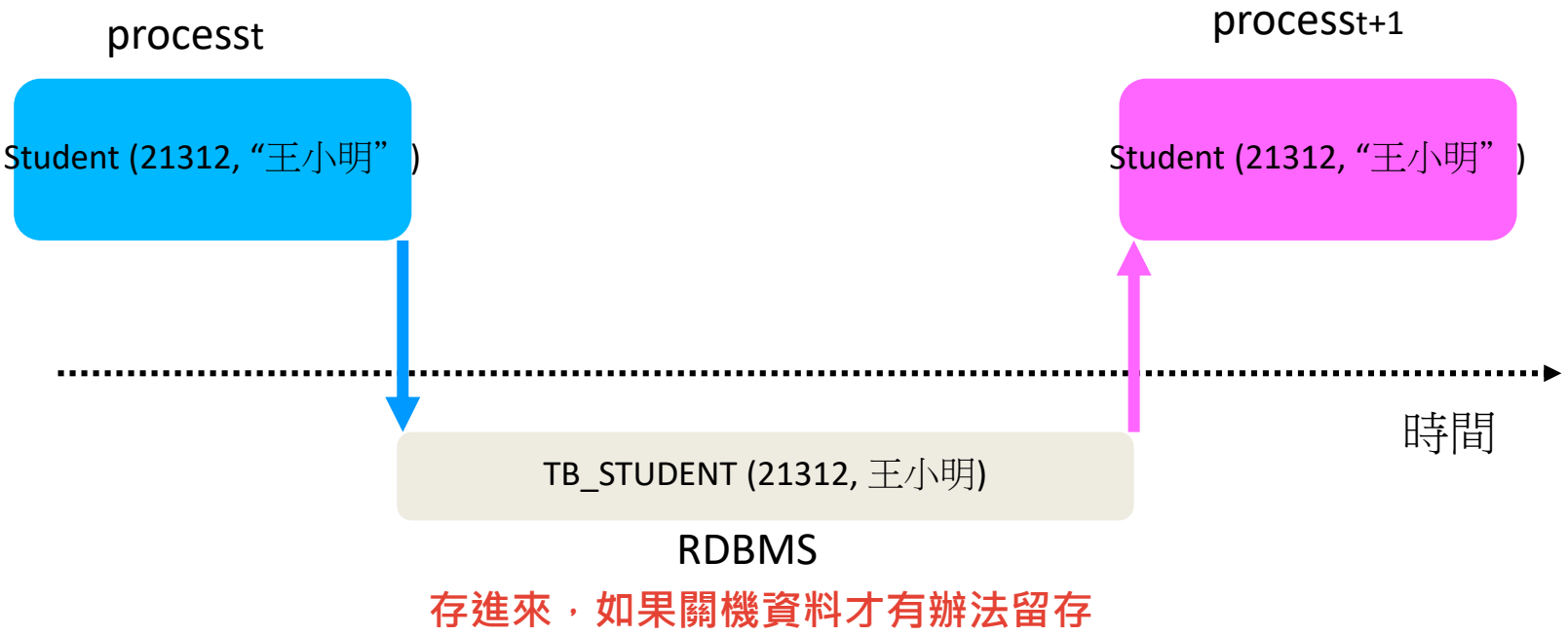
物件的永續性 (Persistence)

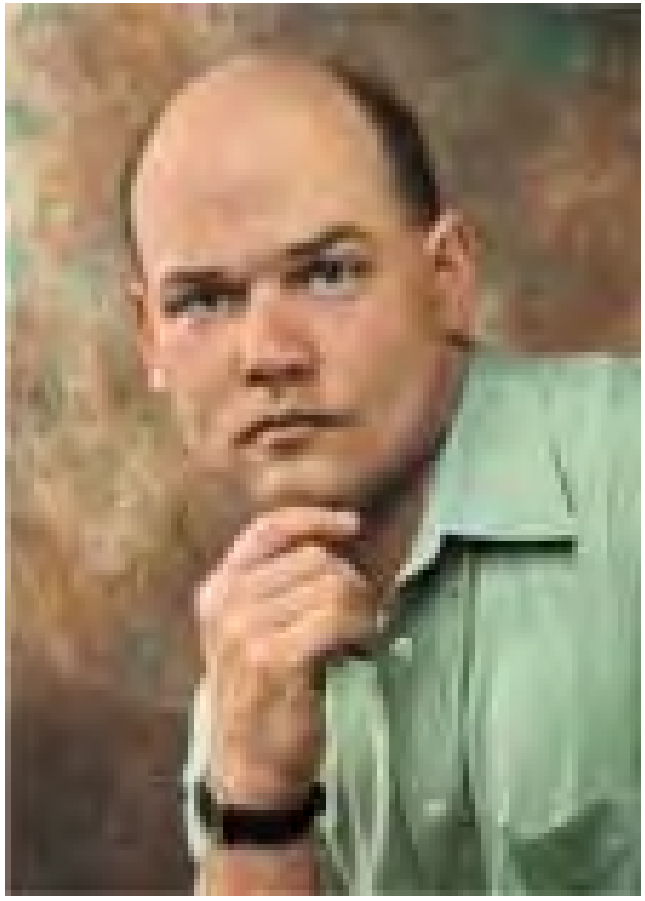
- Java應用程式中的物件存在記憶體(Heap)中

  →JVM重新啟動後，所有物件會連同其狀態一同消失

- 如何長期保存物件狀態?
  - 將物件存在永續性的媒體(如資料庫)上

    →如此一來即使程式重新啟動，物件也可以回復原先的狀態





S.W. Ambler

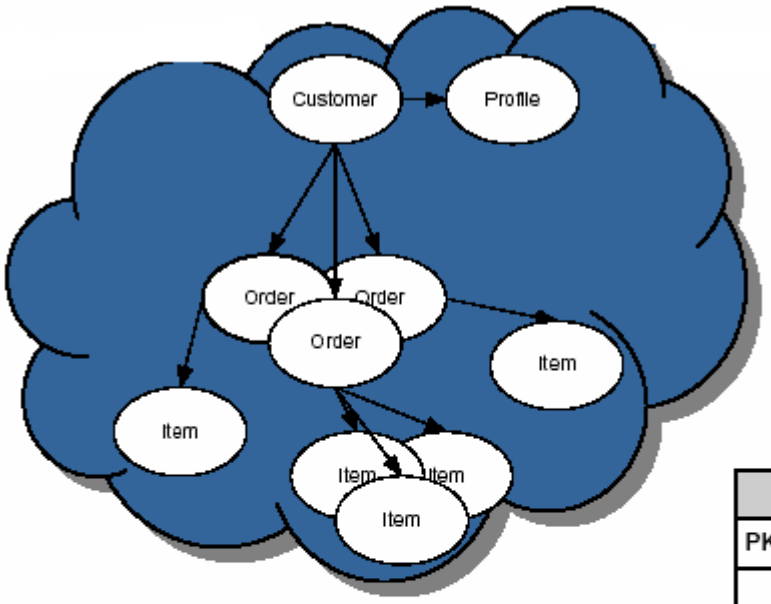
Java的永續性解決方案

- 關聯式資料庫系統(RDBMS)
  - 企業應用程式最常使用的永續性機制

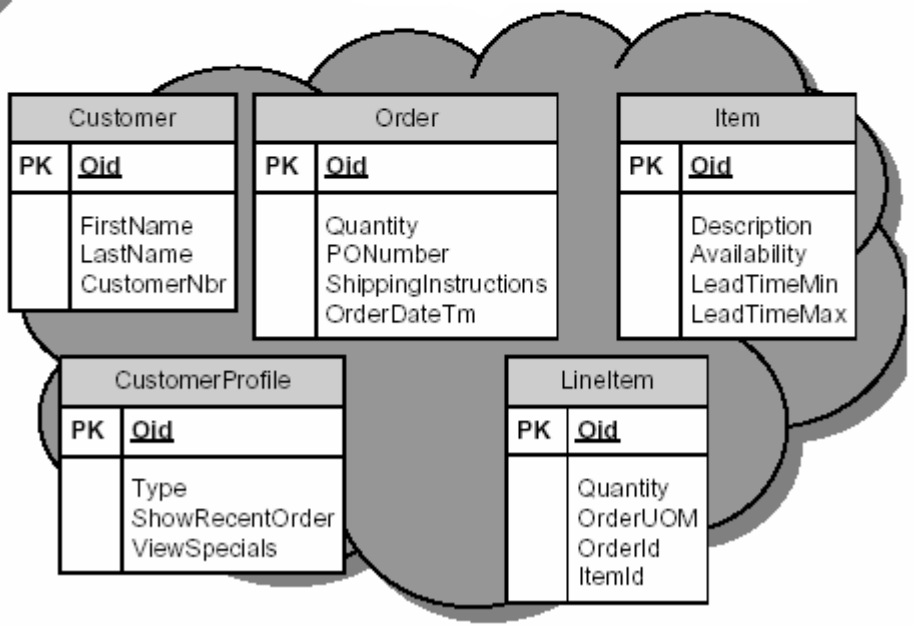
- 問題: 物件導向和關聯式的不相容問題
  - "The Object-Relational Impedance Mismatch Problem"
  - 技術上的不相容
  - 文化上的不相容

S.W. Ambler

ORM: 物件-關連對映



需要人工轉換(因為有例外)



O-R在技術上的不相容

- 下列物件導向技術較難直接對應到RDBMS
  - 關係複雜的物件(Complex objects)
  - 類別繼承體系
  - 覆寫(Overriding)及多載(Overloading)

- 物件導向和關連式資料庫的資料結合方式
  - Objects are traversed through relationships
  - Relational paradigm joins data from tables

Java永續性解決方案的演進

- JDBC時期 (1998 – 2002)
  - 暴力法
  - DAO (封裝JDBC的資料存取程式碼)
  - Apache DBUtils (JDBC Utility Component)
  - iBATIS SQLMap (現為Apache iBATIS)
  - 自己封裝
  - Entity Beans (失敗)

Java永續性解決方案的演進 (續)

永續性

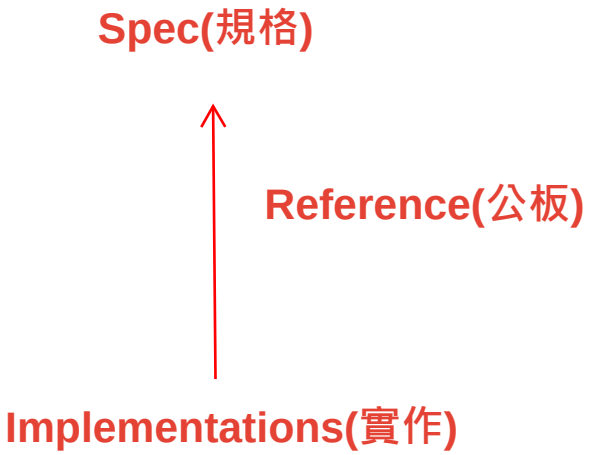
- 自訂Persistence Layer時期 (2002-2005)
  - Apache OJB
  - TopLink
  - Hibernate
  - JDO
  - 自己寫

Gavin King

Java永續性解決方案的演進 (續)

TopLink 轉為開源專案

- 標準Persistence Layer時期 (2006-)
  - Java Persistence API
- Java Persistence API (JPA)
  - POJO-based
    - 容易測試
    - 不再需要DTO (Data Transfer Objects)
  - 可單獨在Java SE中使用
  - 設定檔的預設值可在大部份的場合適用



使用JPA的步驟

- 準備Library
- 設定ORM
  - 開發 POJO Entities
  - 為Entity加上Annotations
- 建立設定檔
  - 在Classpath的最上層建立一個META-INF目錄並在其中建立persistence.xml
- 寫作Client端程式碼
  - 透過EntityManager操作資料

Step1 準備Library

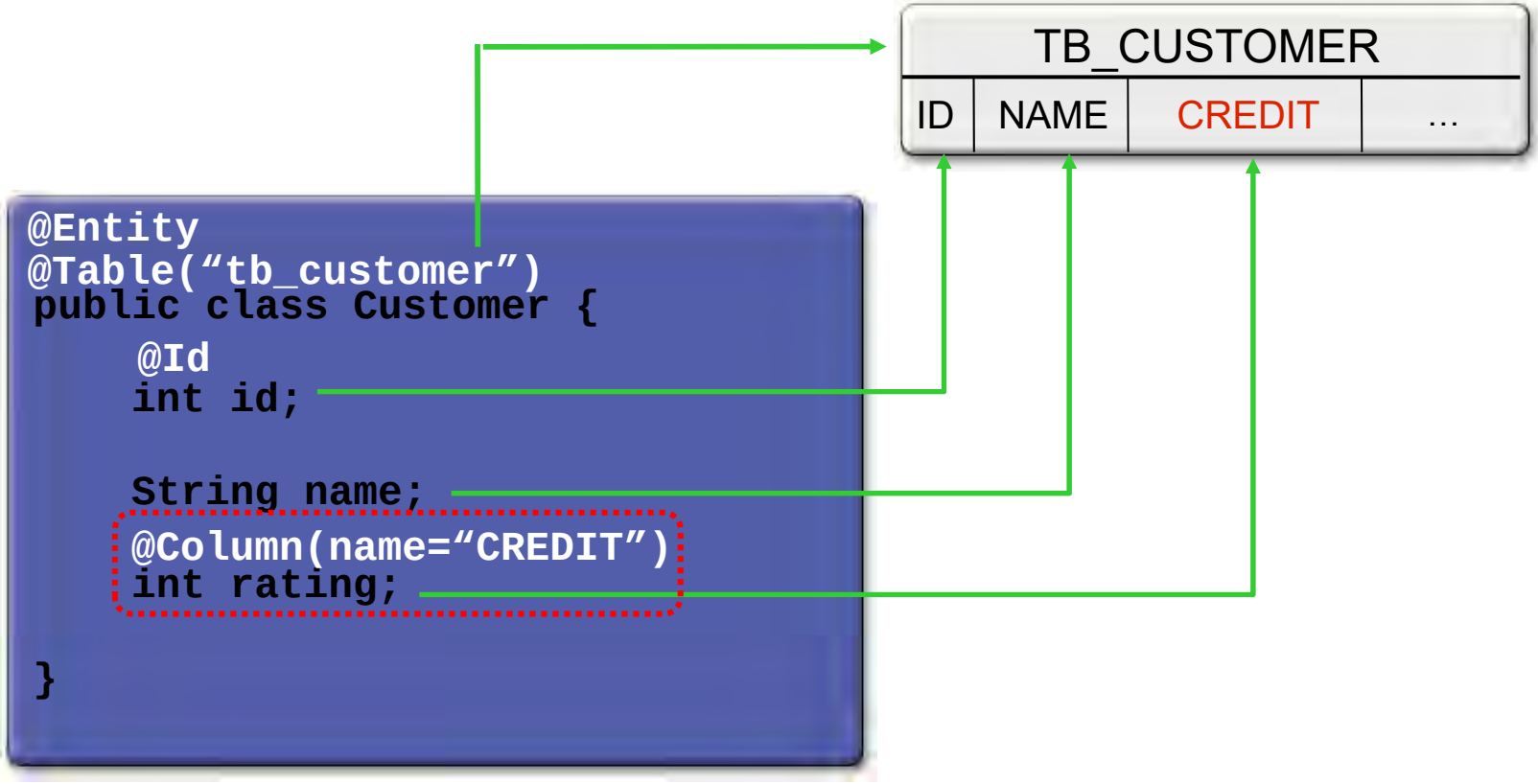
- 將JPA的Library與JDBC驅動程式的jar檔加到專案classpath中
- 常用的JPA實作成品
  - 官方RI: TopLink Essentials
  - Hibernate: Hibernate Core + Hibernate Annotations + Hibernate EntityManager

Step 2 設定ORM


Customer
int id
String name
int rating
...


CUSTOMER
ID
NAME
CREDIT

簡單的ORM對映



預設會直接將field名稱對映到column名稱

→只有在field名稱不同時才需要特別設定

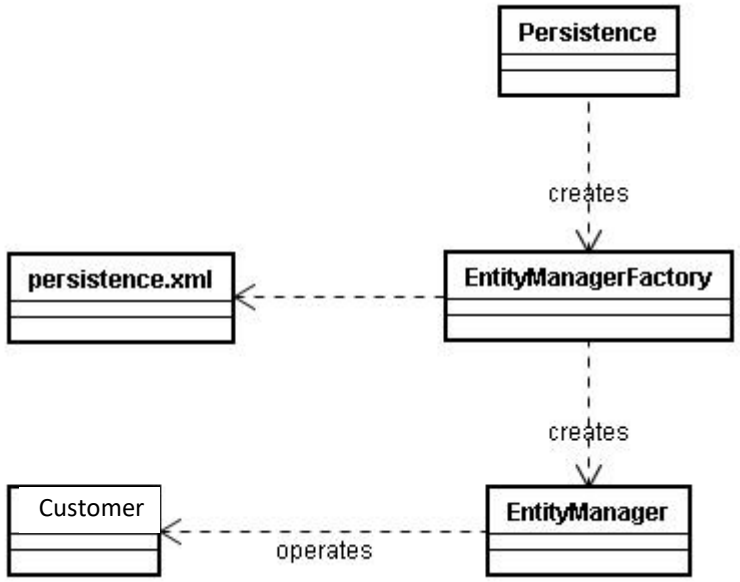
Step 3 建立設定檔

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    version="1.0">
   <persistence-unit name="CustomerService">
      <class>Customer</class> 使用類別全稱
      <properties>
         <property name="toplink.jdbc.driver" value=" JDBC驅動程式全名"/>
         <property name="toplink.jdbc.url" value="資料庫的url"/>
         <property name="toplink.jdbc.user" value="使用者名稱"/>
         <property name="toplink.jdbc.password" value="使用者密碼"/>
         <property name="toplink.logging.level" value="FINE"/>
      </properties>
   </persistence-unit>
</persistence>
```

依使用的實作品有所不同

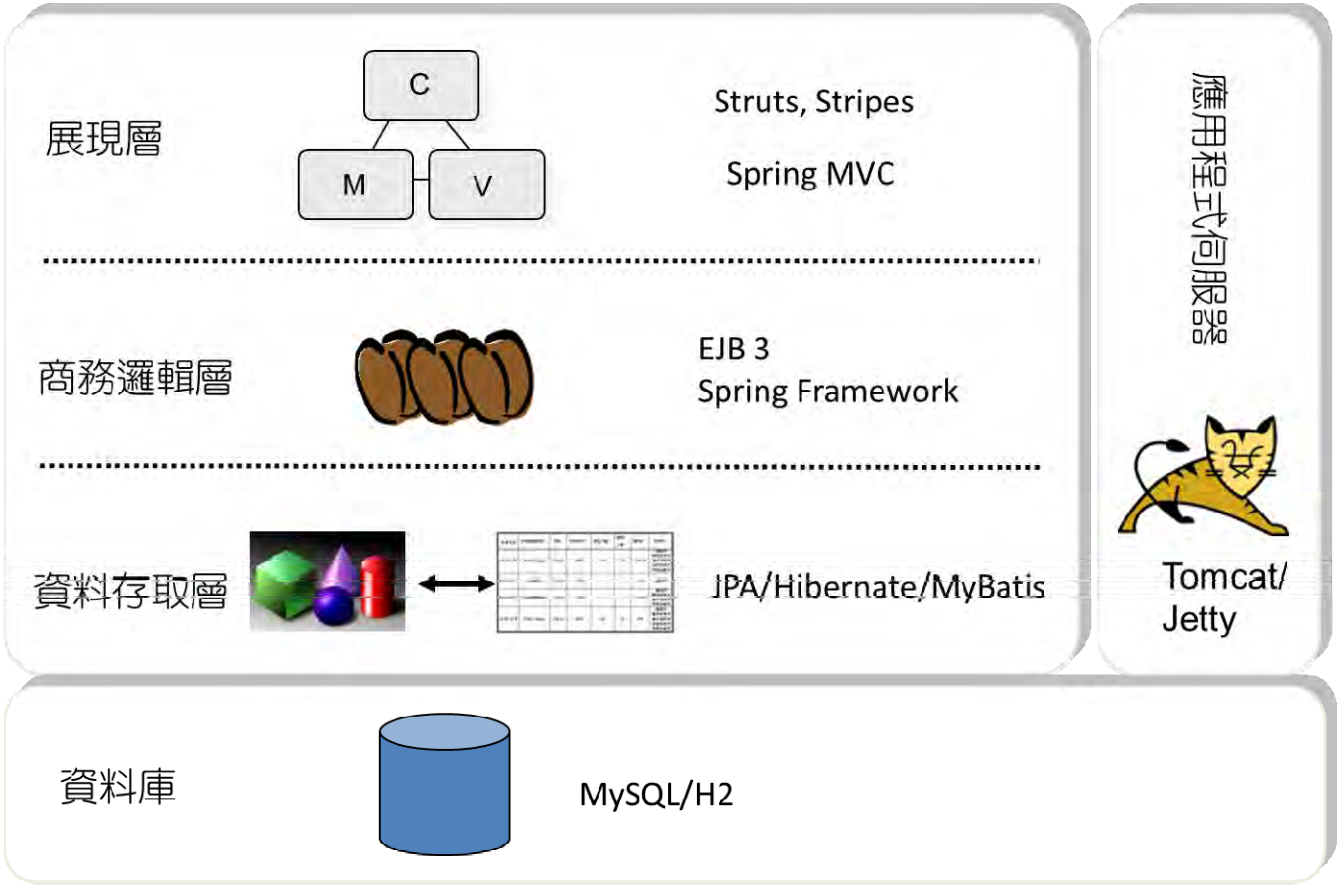
Step 4 寫作Client 程式

```
Public static void main(String args[]){
        // 取得EntityManager的前置作業
        EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("CustomerService");
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        Customer c = new Customer();
        … // 填入資料
        em.persist(c);
        em.getTransaction().commit();
        em.close();
        emf.close();
}
```



相關設計案例

- JPetstore



相關設計案例

- 政大校務系統

Q & A