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

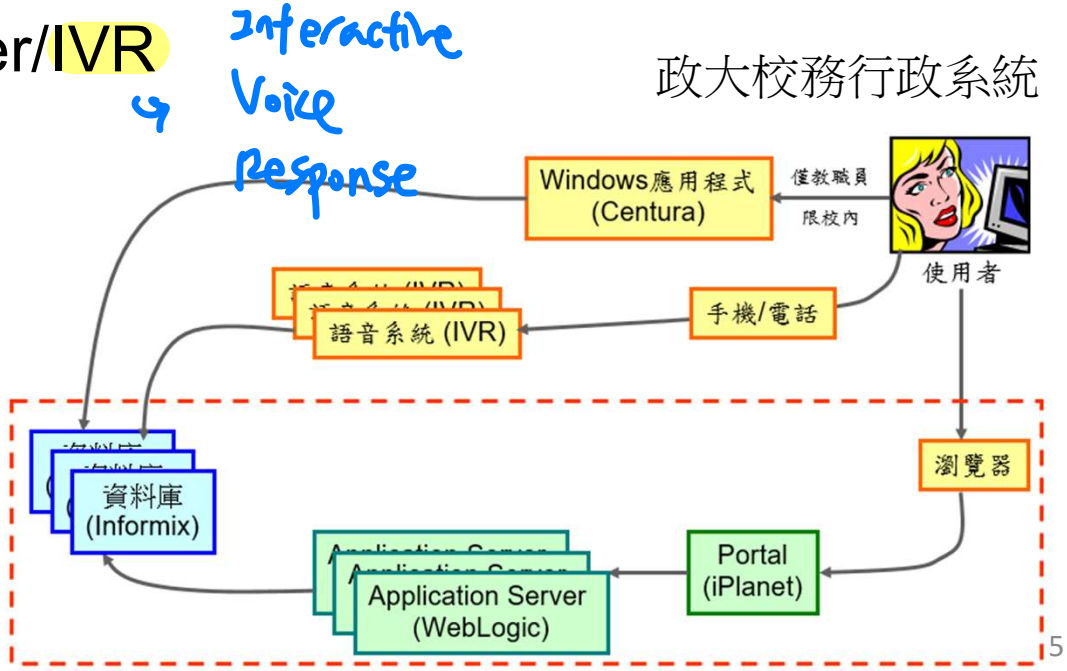
- Enterprise applications
  - Payroll
  - Patient records
  - Shipping tracking
  - Cost analysis
  - Credit scoring
  - Insurance
  - Supply chain
  - Accounting
  - Customer service
  - Foreign exchange trading
- NOT Enterprise applications
  - Automobile fuel injection,
  - Word processors,
  - Elevator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

企業應用程式特性

- Persistent
  - 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
- A lot of data
  - 中型系統: 1G 以上資料需要保存
  - 一般存於RDBMS
  - 更早期系統存於IBM VSAM/ISAM
    - Indexed Sequential Access Method (older)
    - Virtual Storage Access Method (more modern)

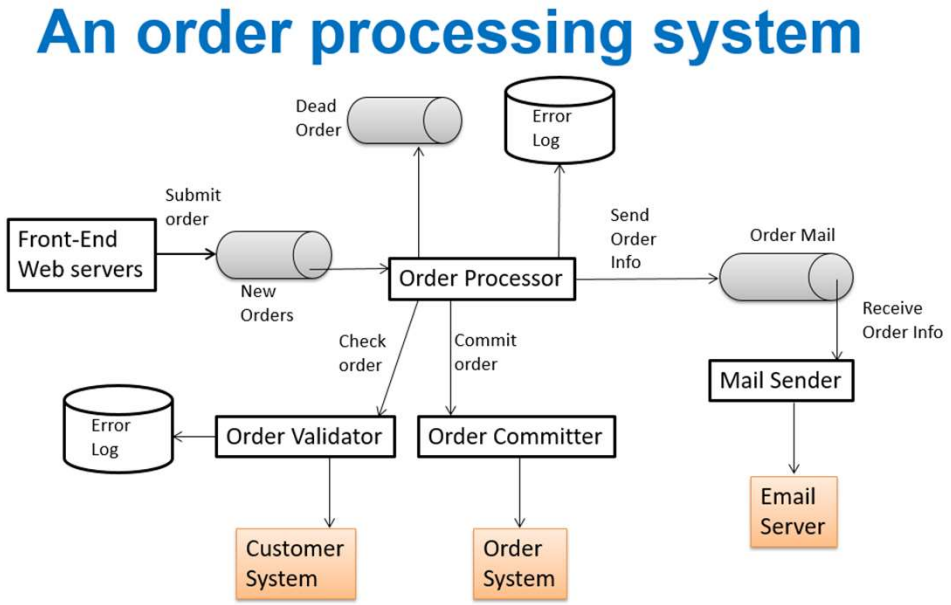
企業應用程式特性

- Services access data concurrently
  - Race Condition: 同時寫入同筆資料會出現問題
  - Transactions概念變重要
- Multiple UI to access the services
  - Web/Client-Server/IVR

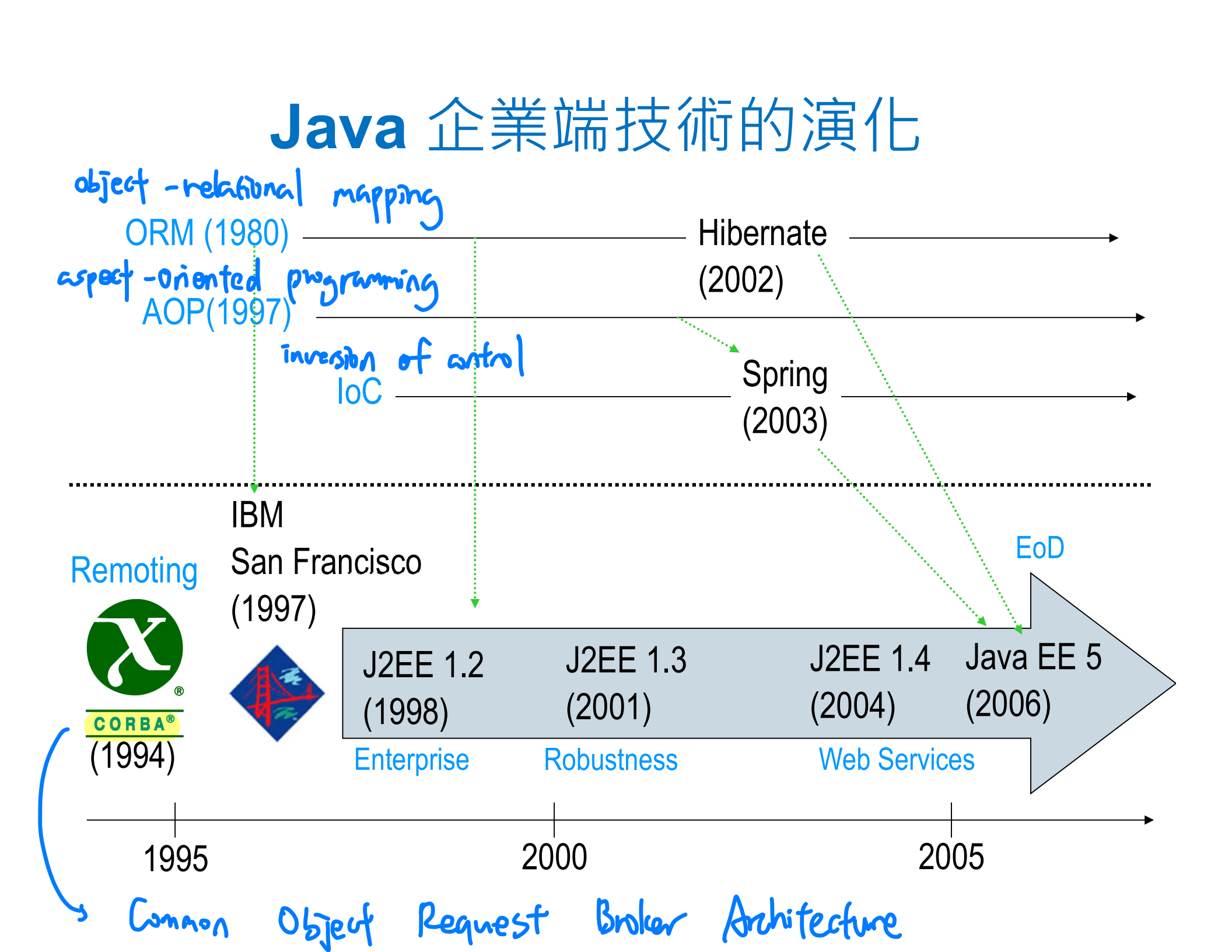


企業應用程式特性

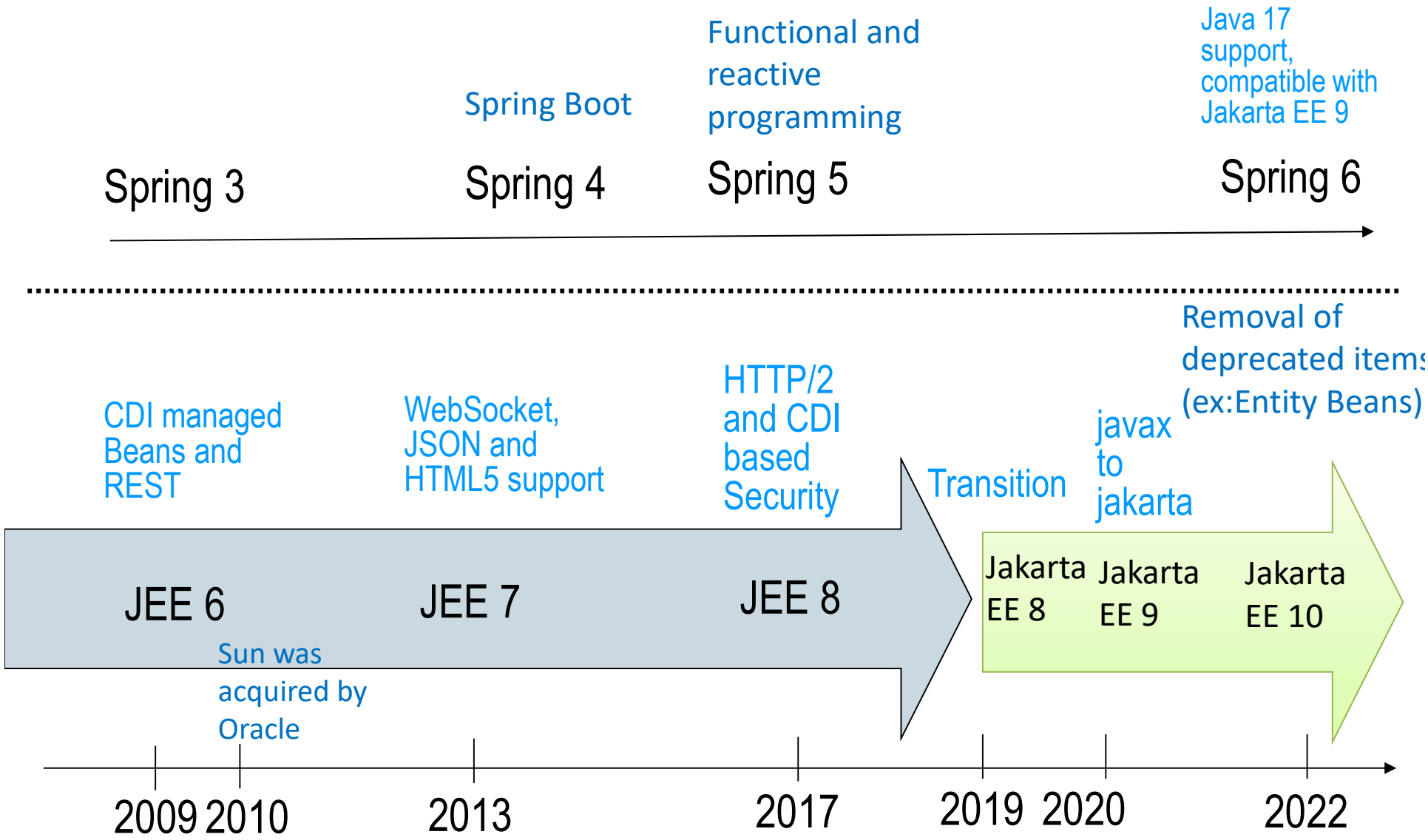
- Integrate with other enterprise applications
  - Enterprise applications rarely live on an island
  - Typically via Web Services or Messaging Systems



Java 企業端技術的演化



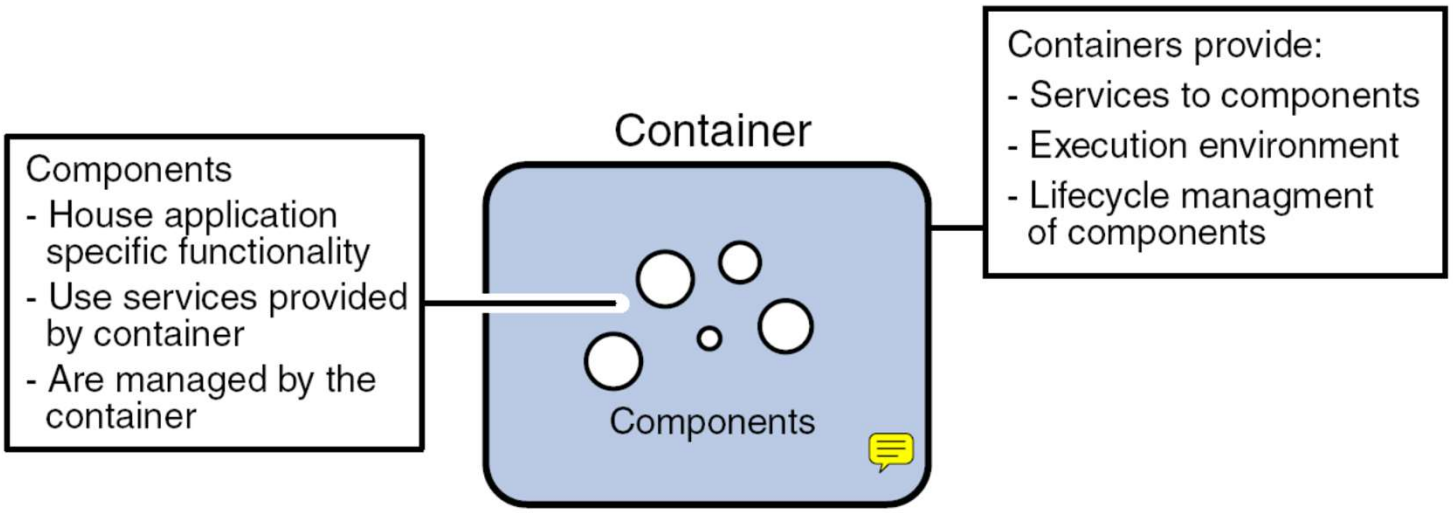
Java 企業端技術的演化



元件

容器

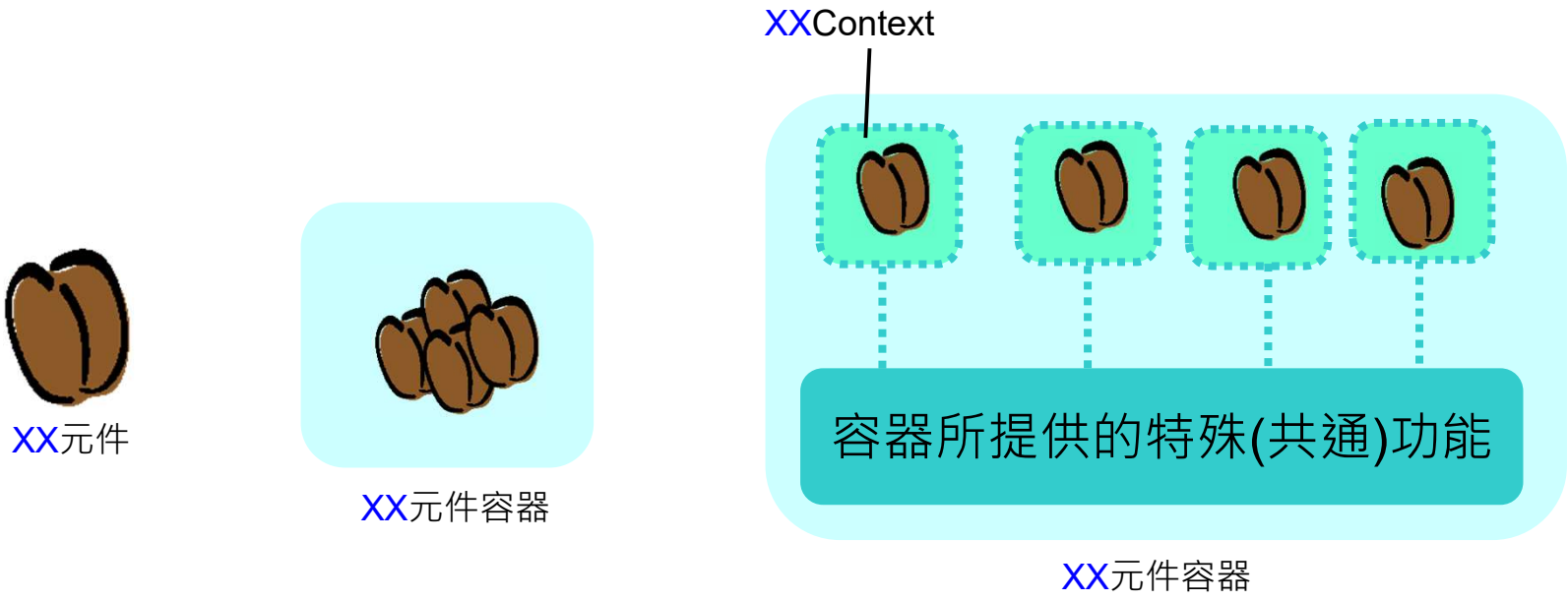
容器/元件架構



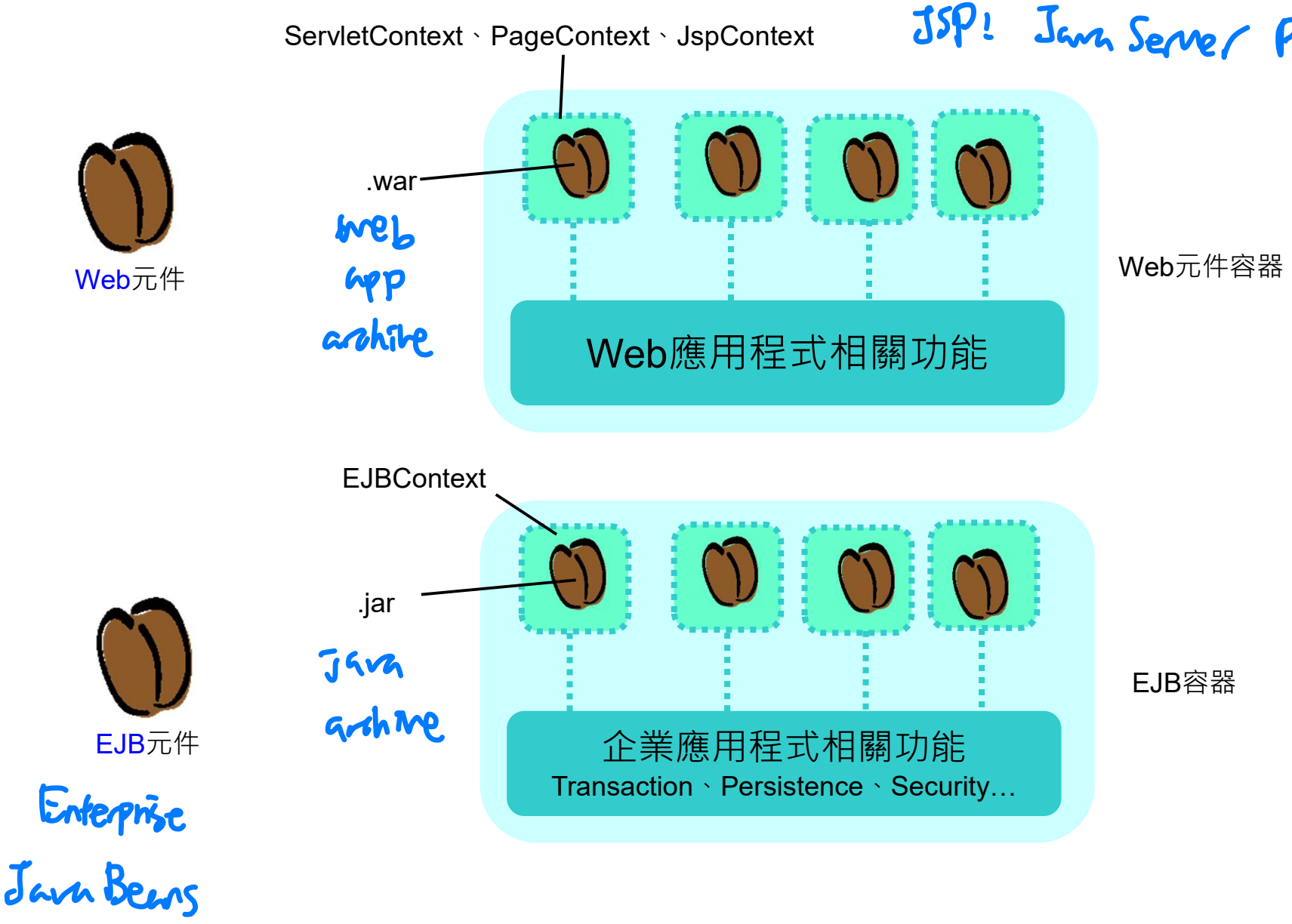
元件/容器溝通的媒介稱為Context

早期: 一個容器，很多元件; 現在: 一個容器，一個元件

元件-容器模型

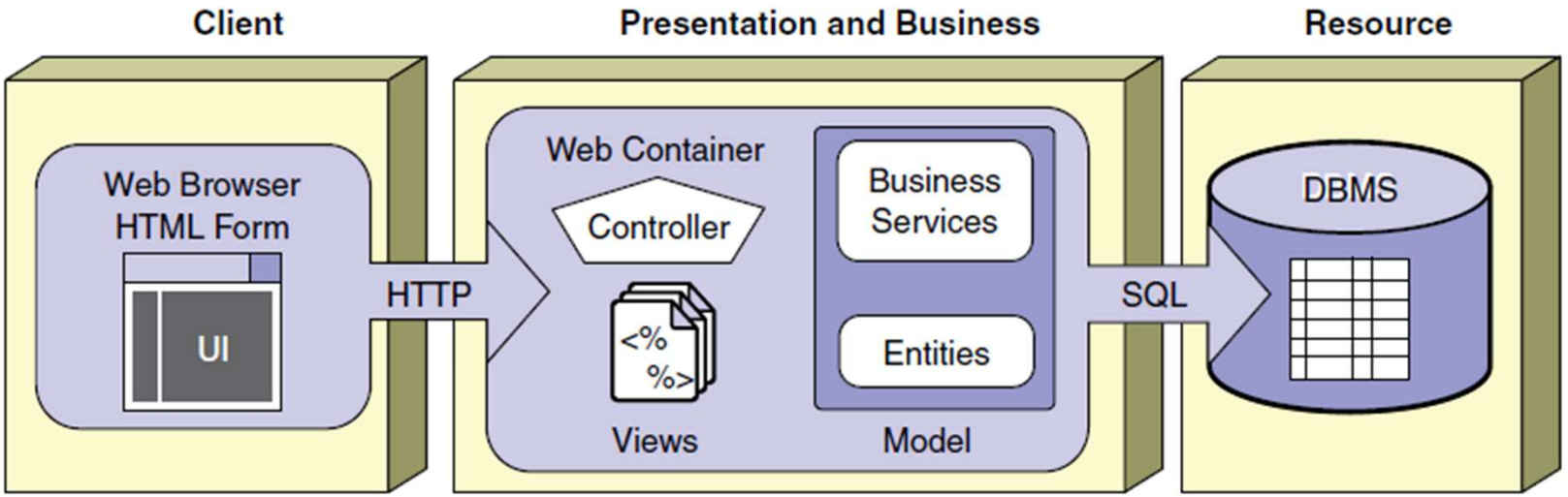


常見的元件模型


ServletContext、PageContext、JspContext
JSP! Java Server Pages
.war
web app archive
Web元件
Web元件容器
Web應用程式相關功能
EJBContext
.jar
Java archive
EJB元件
EJB容器
企業應用程式相關功能
Transaction、Persistence、Security…
Enterprise Java Beans

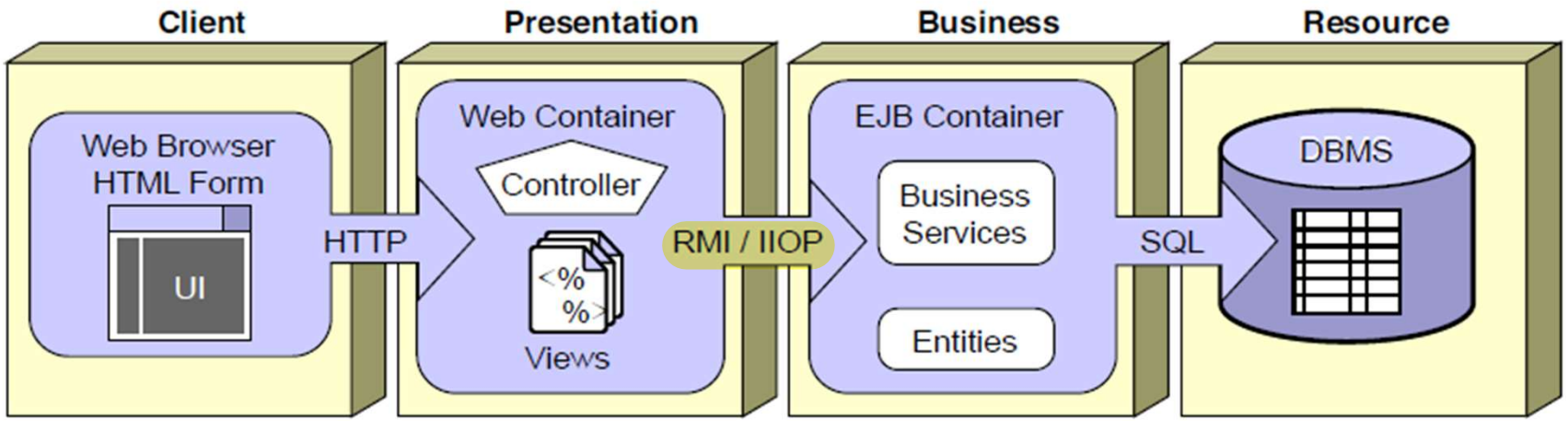
企業應用程式架構

- Web centric



企業應用程式架構

- Middleware centric
  - Ex: Java EE containers



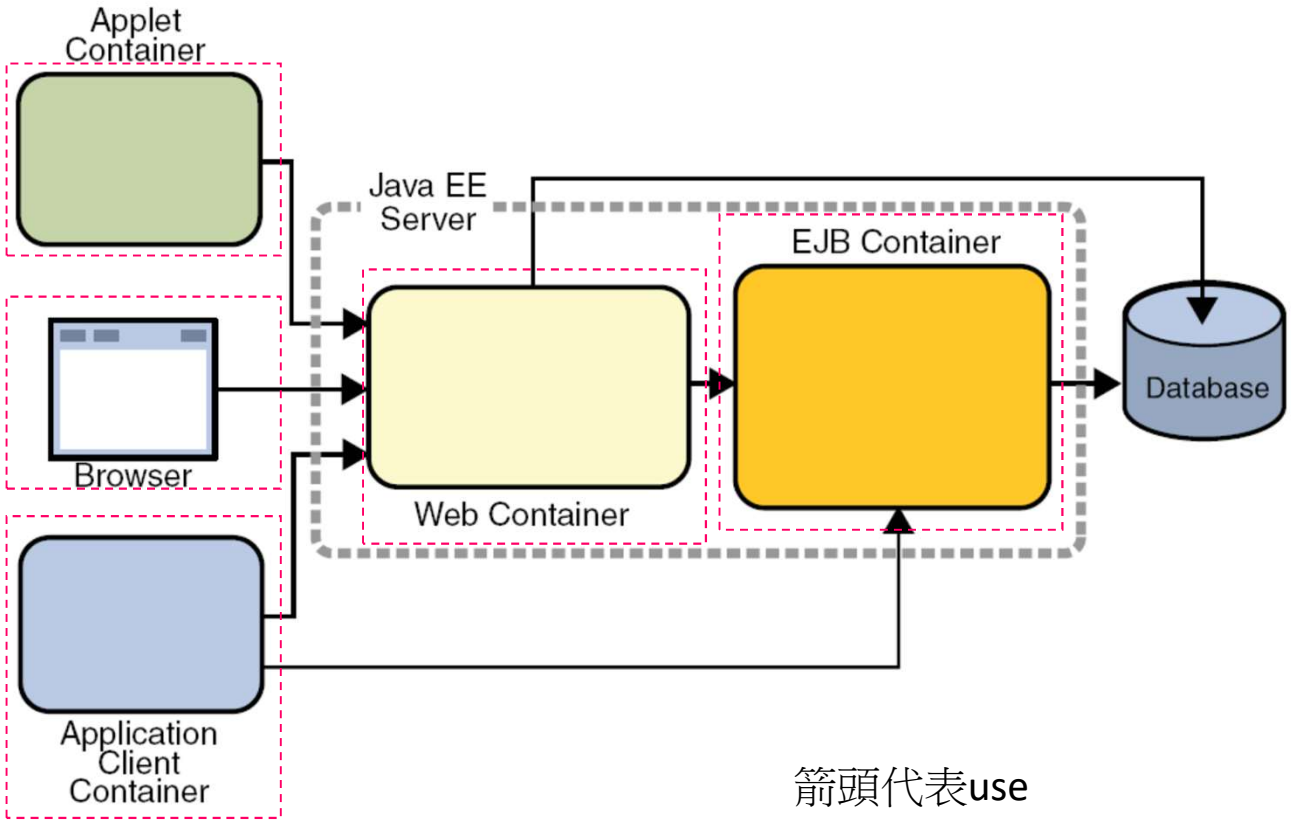
RMI : Remote Method Invocation

IIOP: Internet Inter-ORB Protocol

object Request Broker

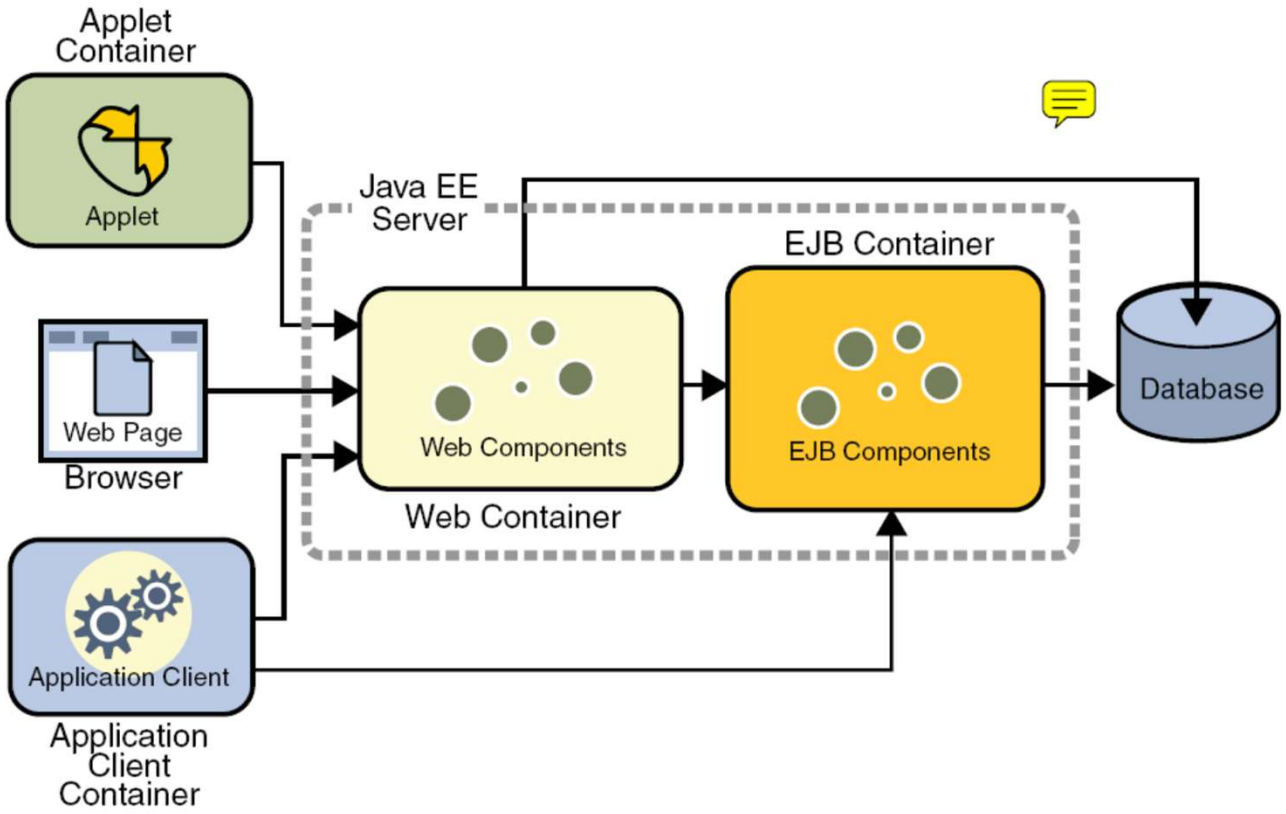
Java EE 的容器

這些容器的元件各是什麼?



箭頭代表use

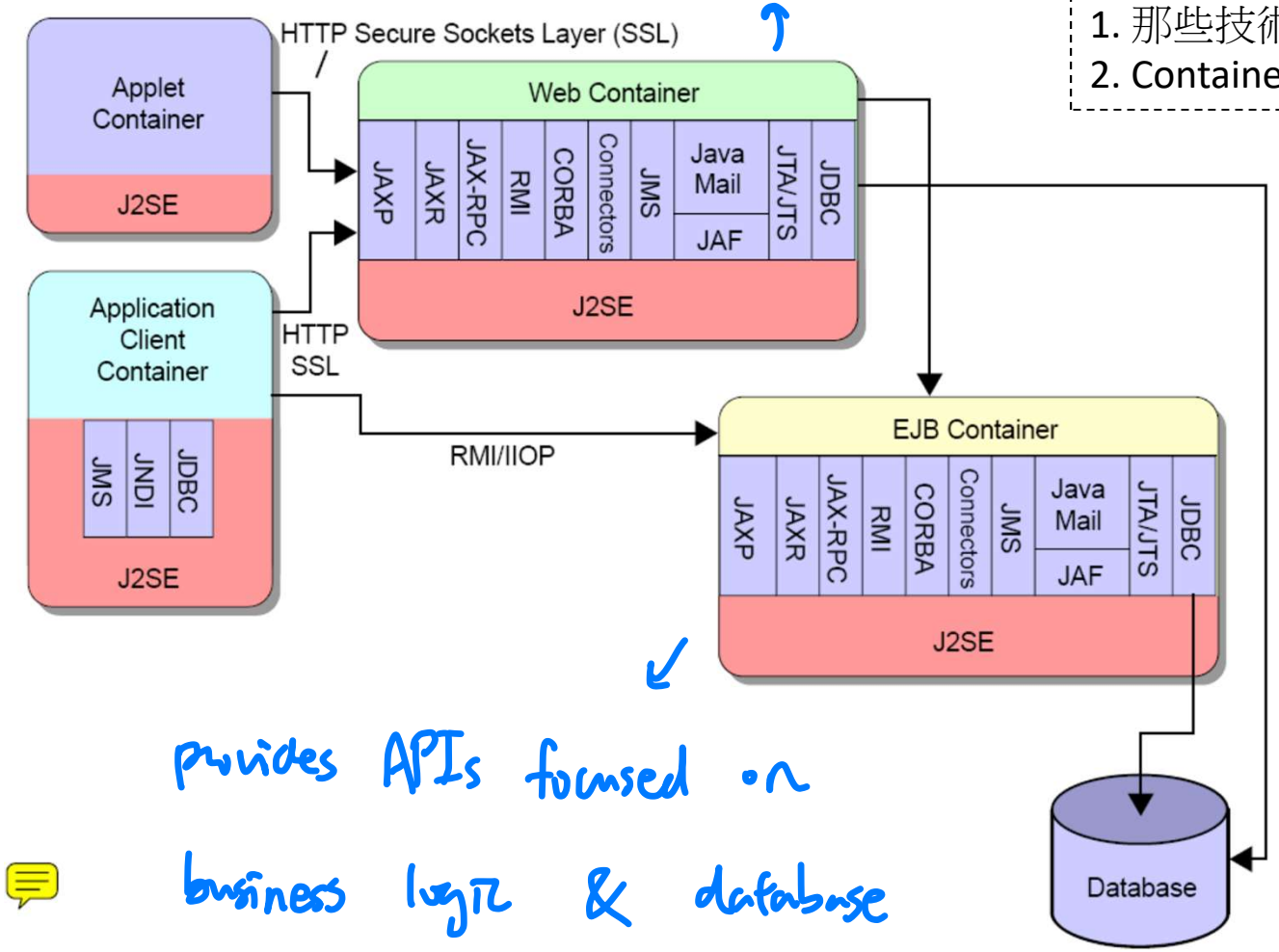
Java EE 的元件


Applet Container
Applet
Java EE Server
EJB Container
Web Page
Browser
Web Components
Web Container
EJB Components
Database
Application Client
Application Client Container

API Services

provides web-related APIs

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



* J2SE : Java 2 Platform, Standard Edition

provides APIs focused on business logic & database interaction

Classical Enterprise Application Development 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→ORM
  - Logic→SessionBean
  - Async→MDB
  - Client
- 佈署階段
  - Assemble / Packaging / Deploy

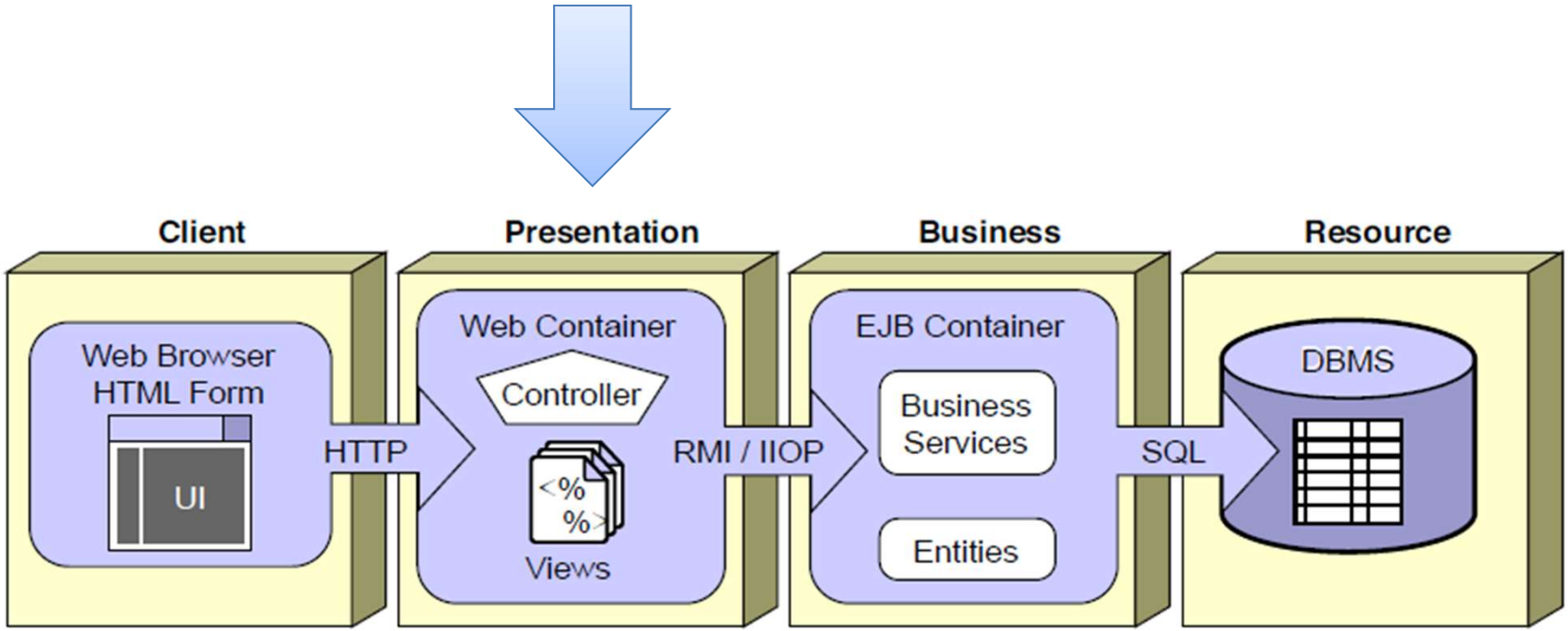
Unified Modeling Language

entity class —map→ database table

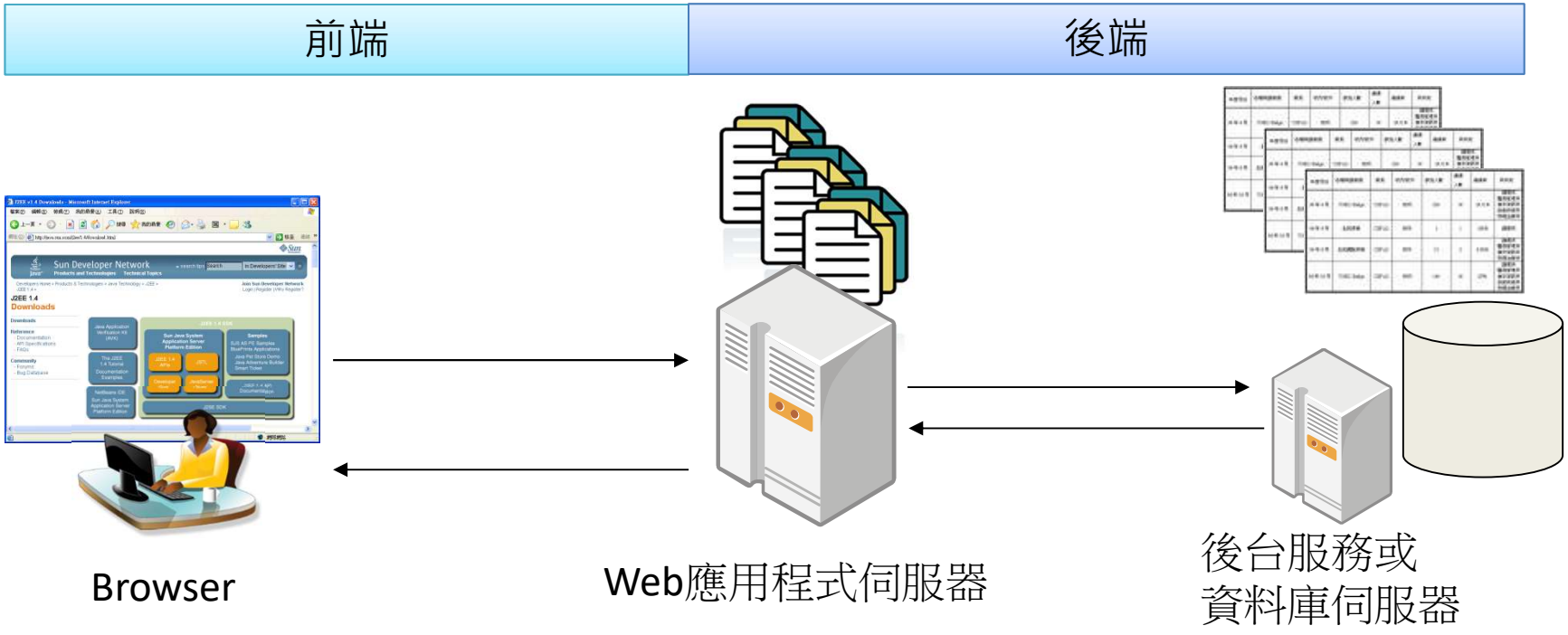
business logic —encapsulate→ enterprise applications

message-driven beans : handle message asynchronously

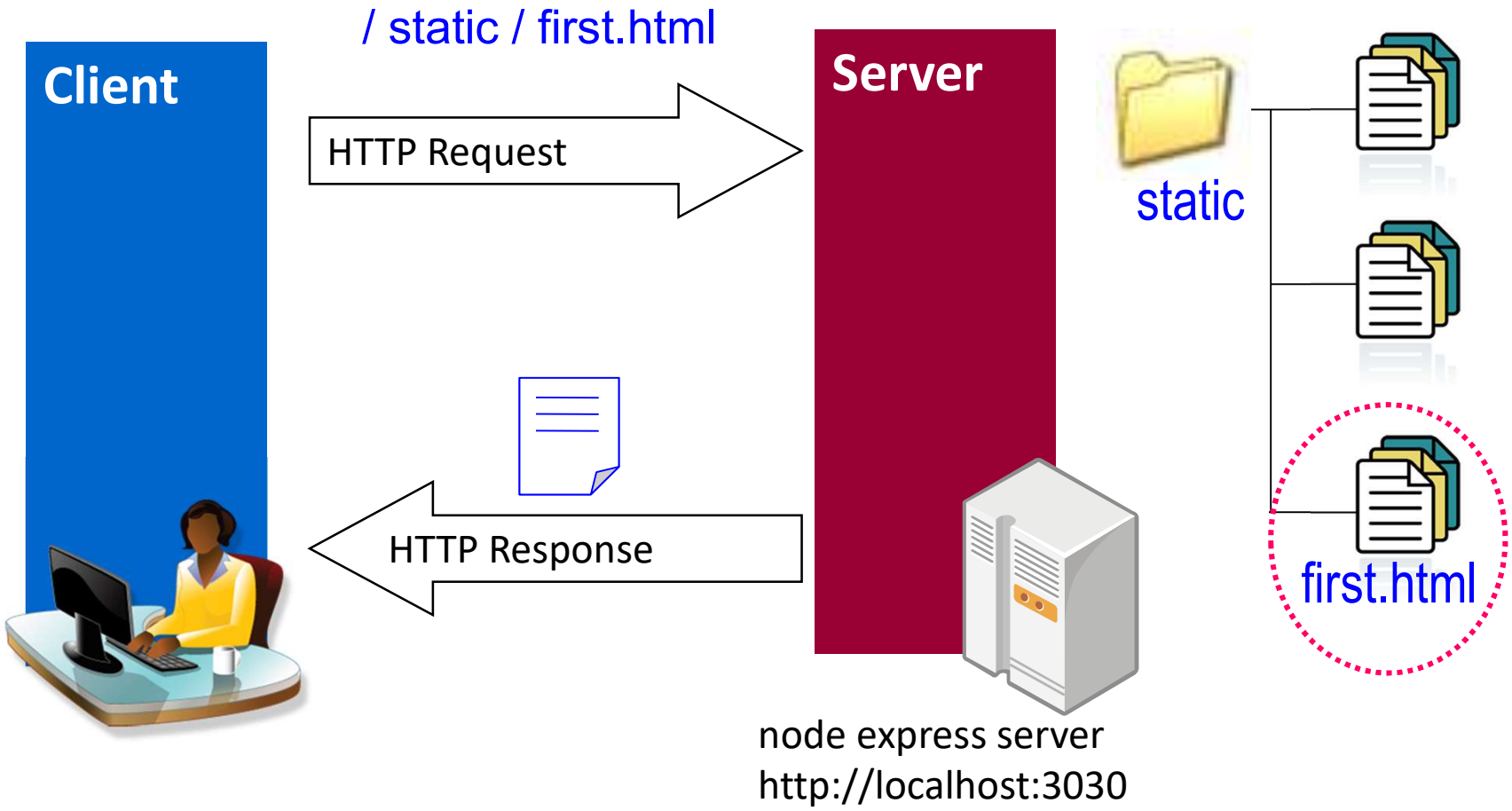
Web應用程式架構


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

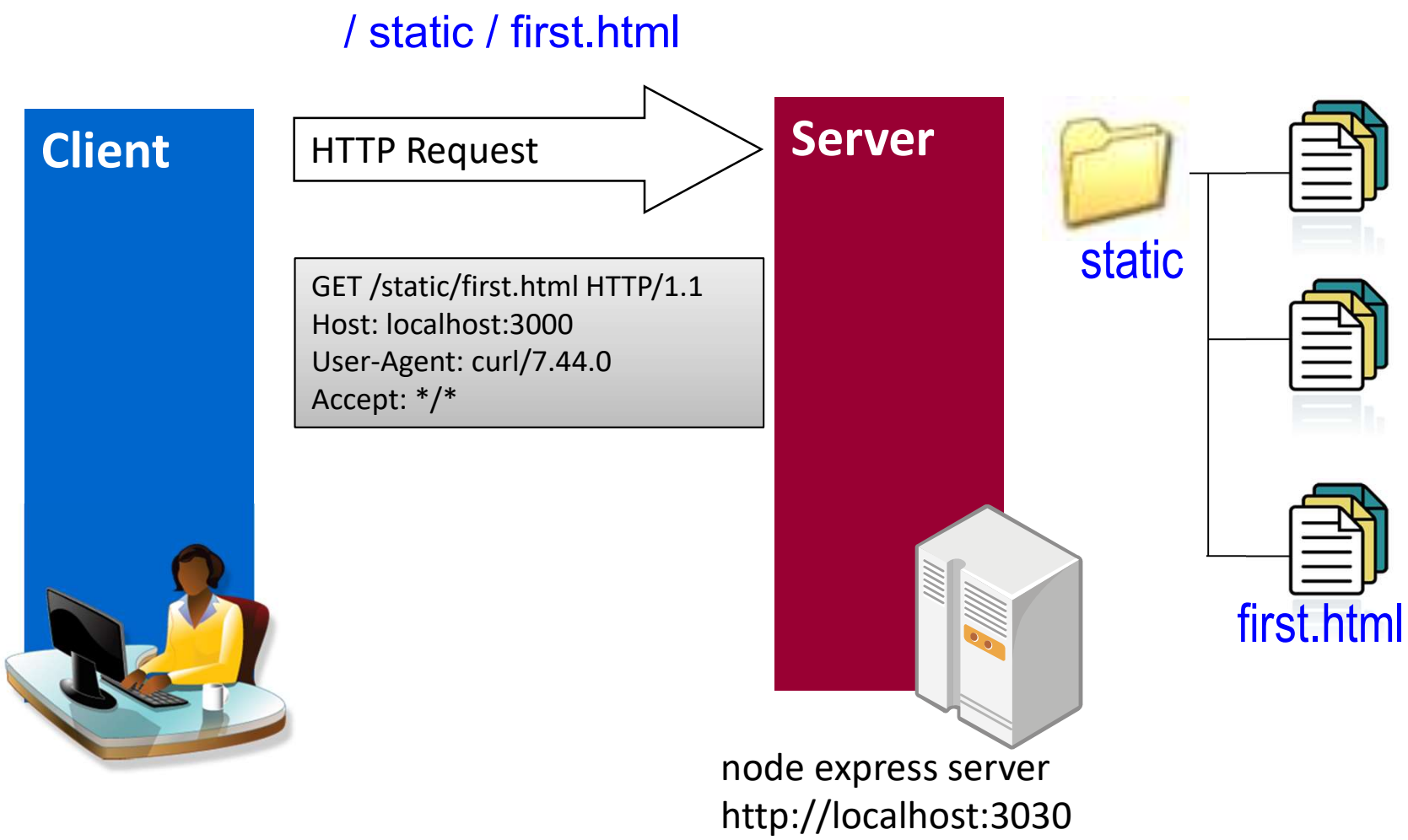
Web應用程式架構


前端
後端
Sun Developer Network
J2EE 1.4
Downloads
Browser
Web應用程式伺服器
後台服務或
資料庫伺服器
如何跟User互動
如何跟後台服務/資料庫互動

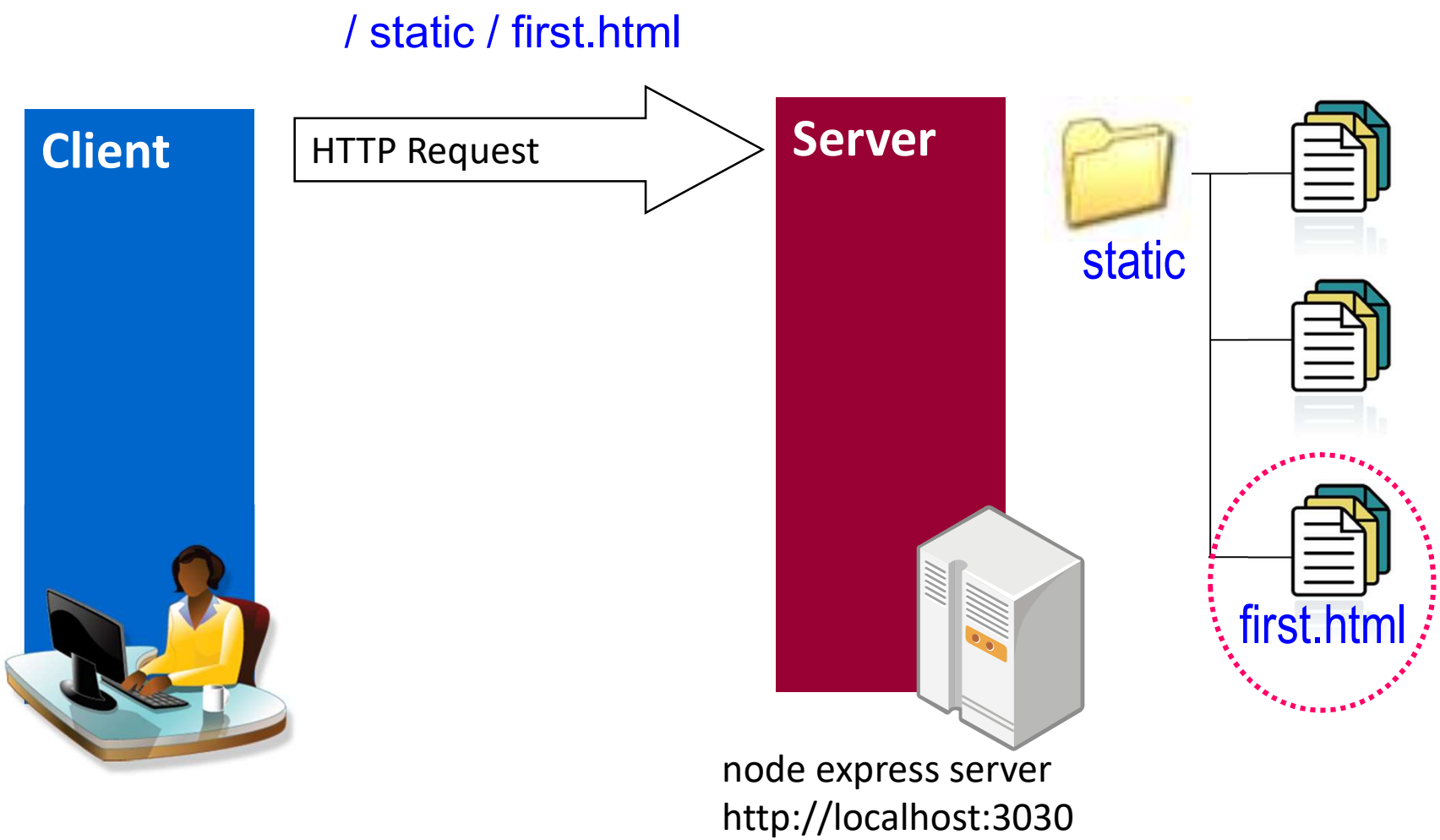
傳送靜態文件


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

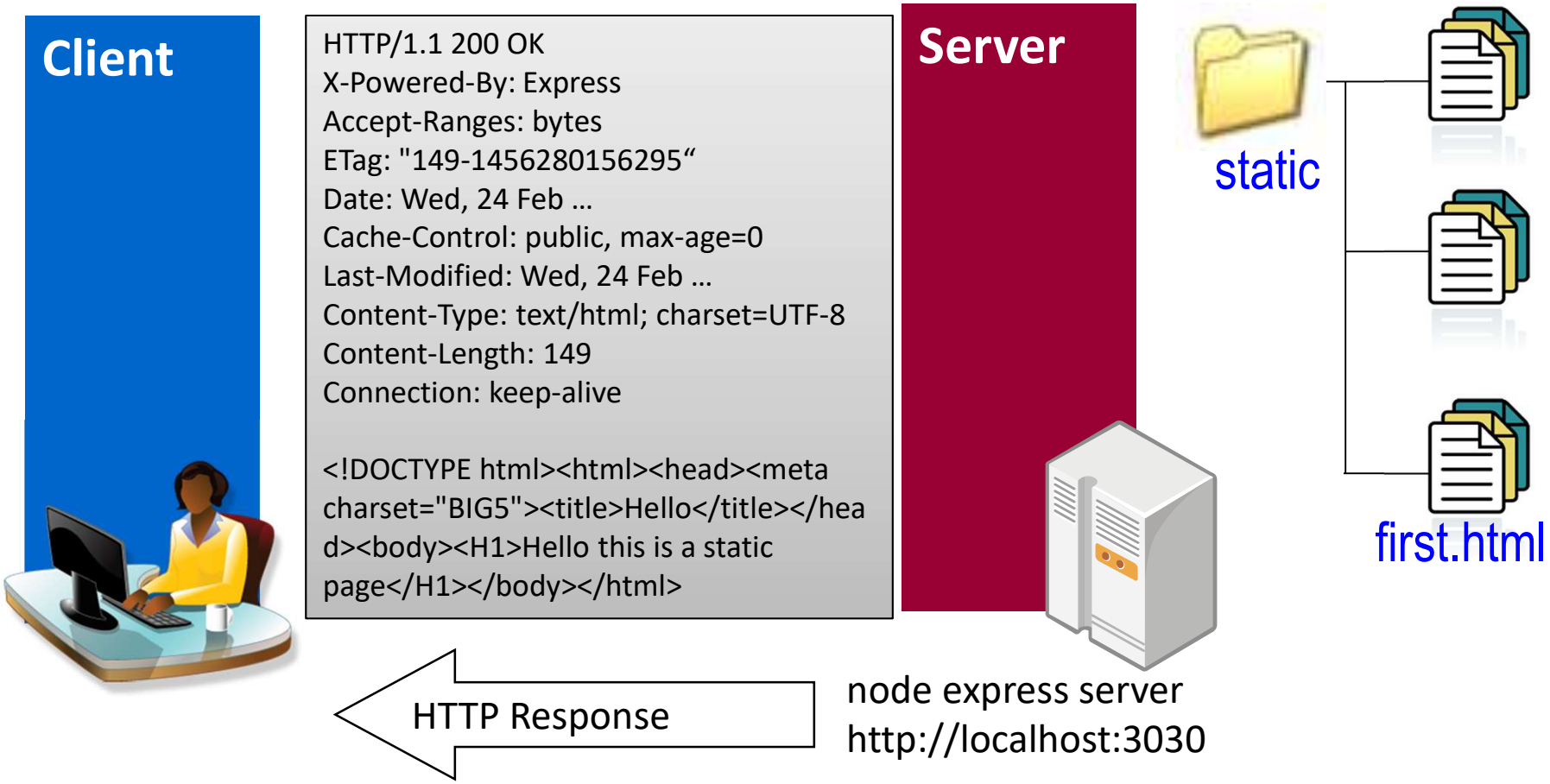
傳送靜態文件


/ 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

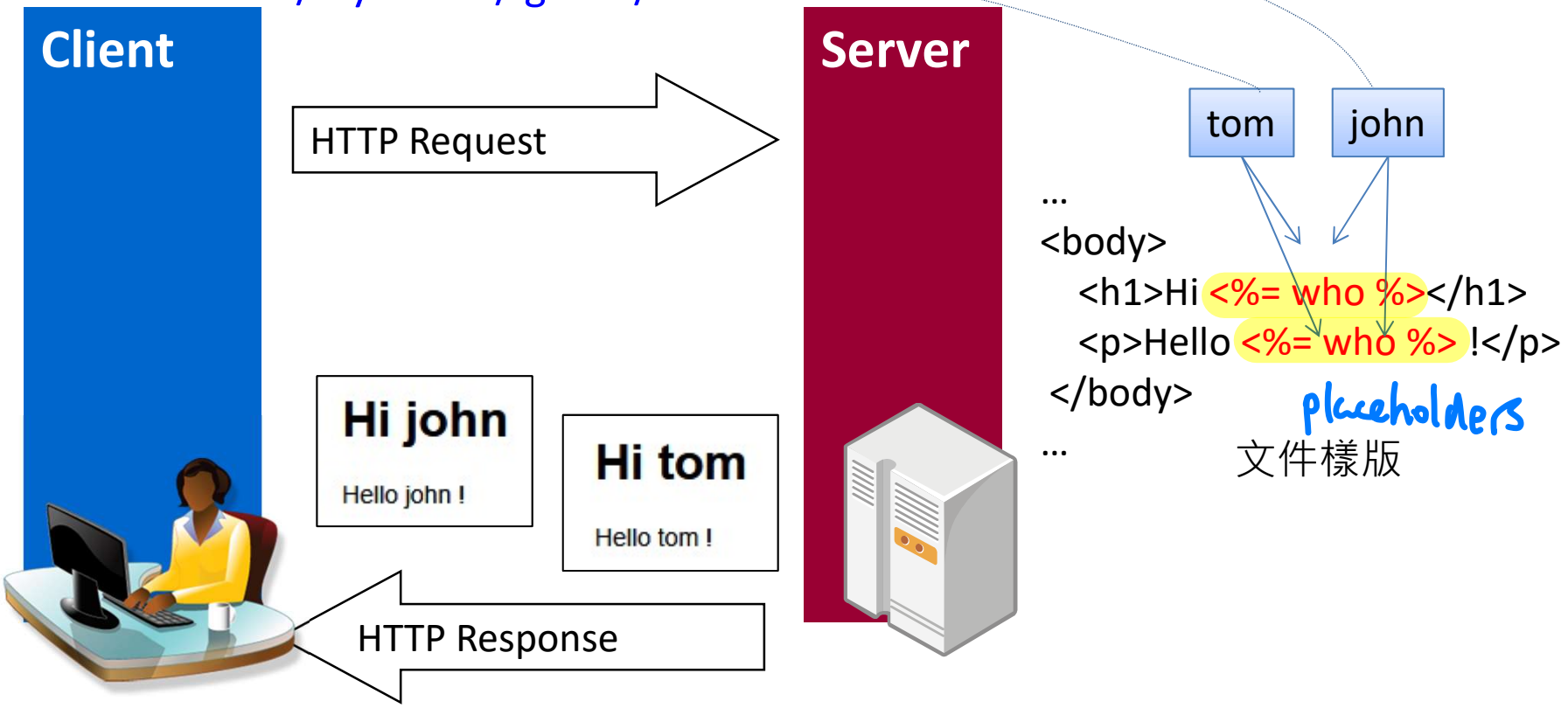
JavaServer Pages

隨需合成文件: JSP/PHP

Hypertext Preprocessor

/ dynamic / greet / john

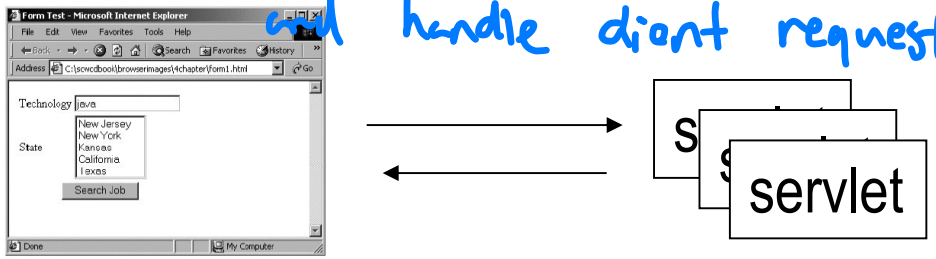
/ dynamic / greet / tom



CGI風格與頁面樣版風格

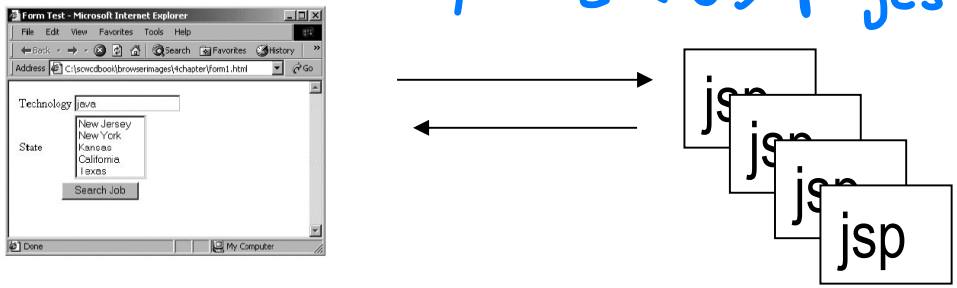
Java programs that run on a server and handle client requests.

my JAVA with HTML




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

A technology that allows creation of dynamic web pages




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

including JAVA code within HTML using <% %>

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

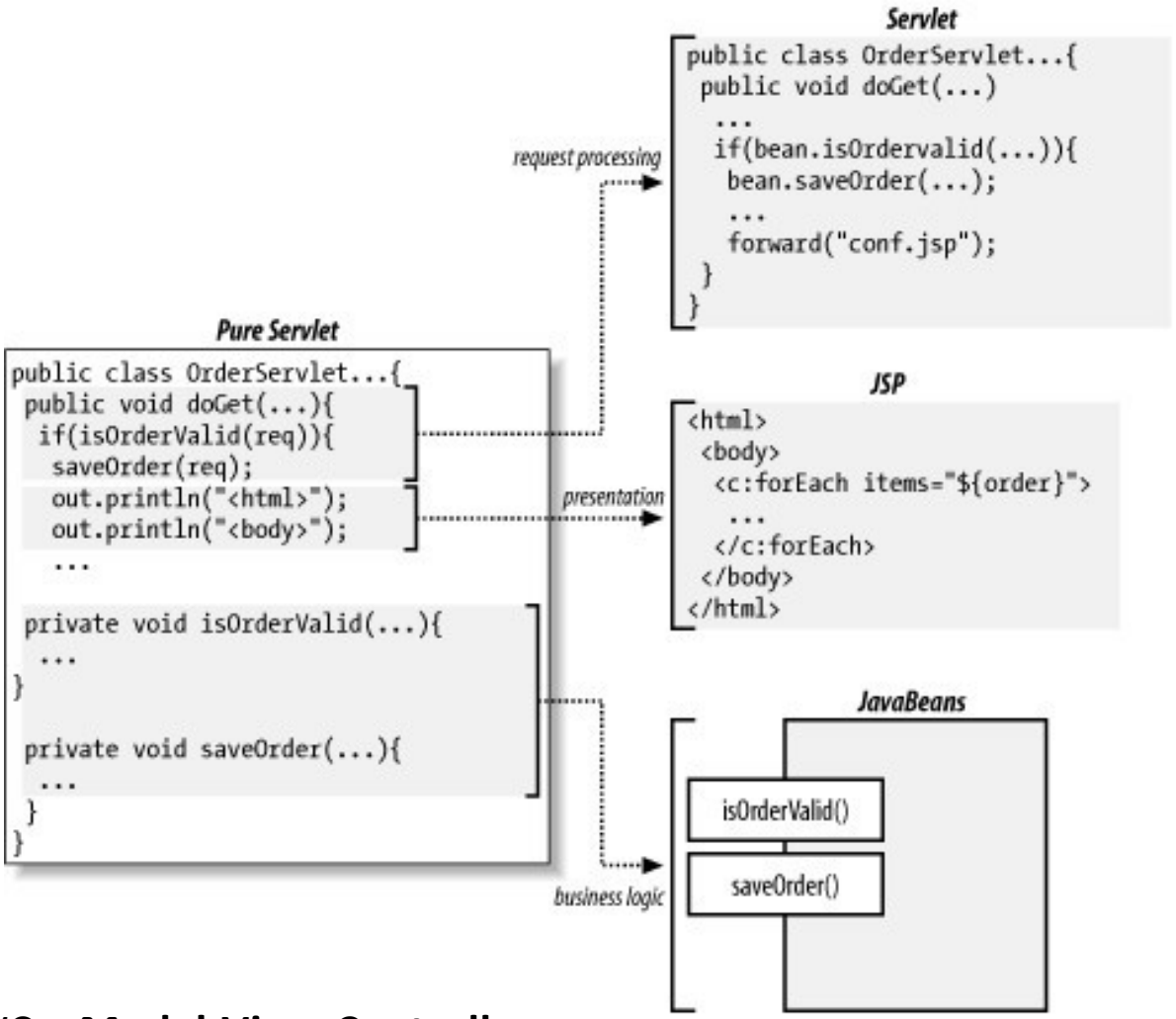
```
out.println("<body>");

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\">");

  …
```

Servlet

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



controller

C 解讀使用者的資料，決定網頁流程

view

V Html網頁

model

M 商務邏輯、資料

MVC = **Model-View-Controller**

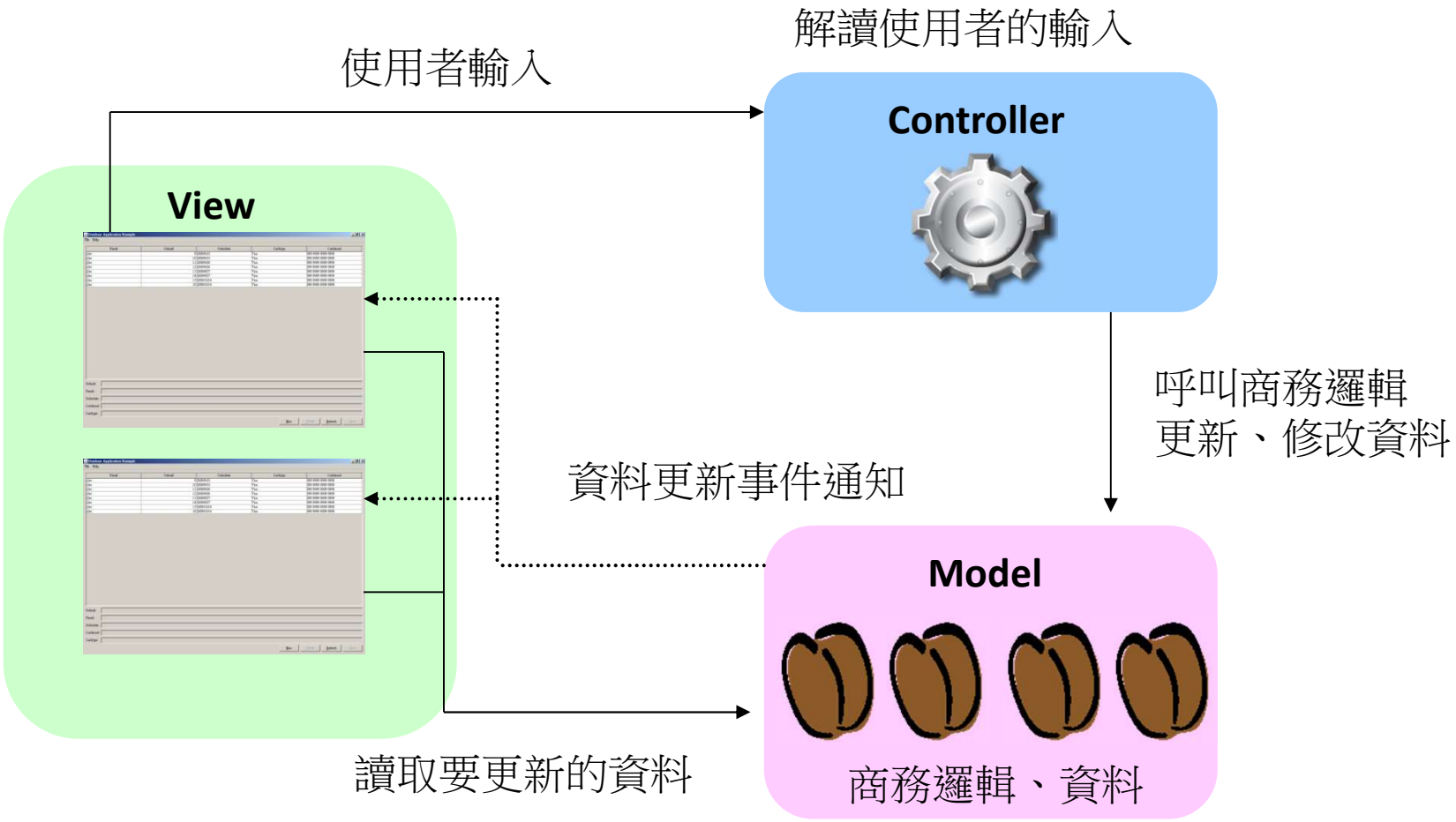
MVC架構

- 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

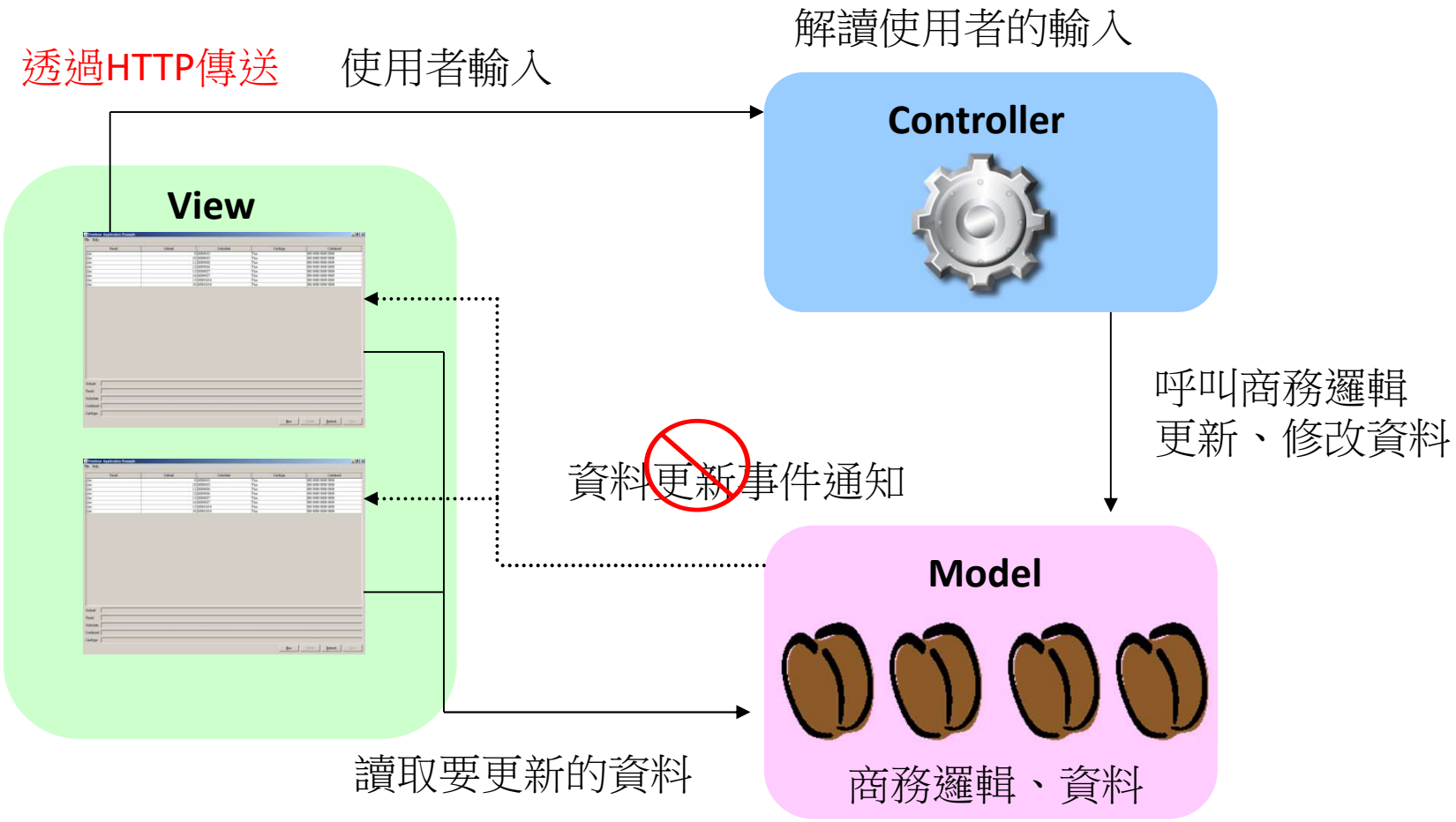
使用MVC的好處

- 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

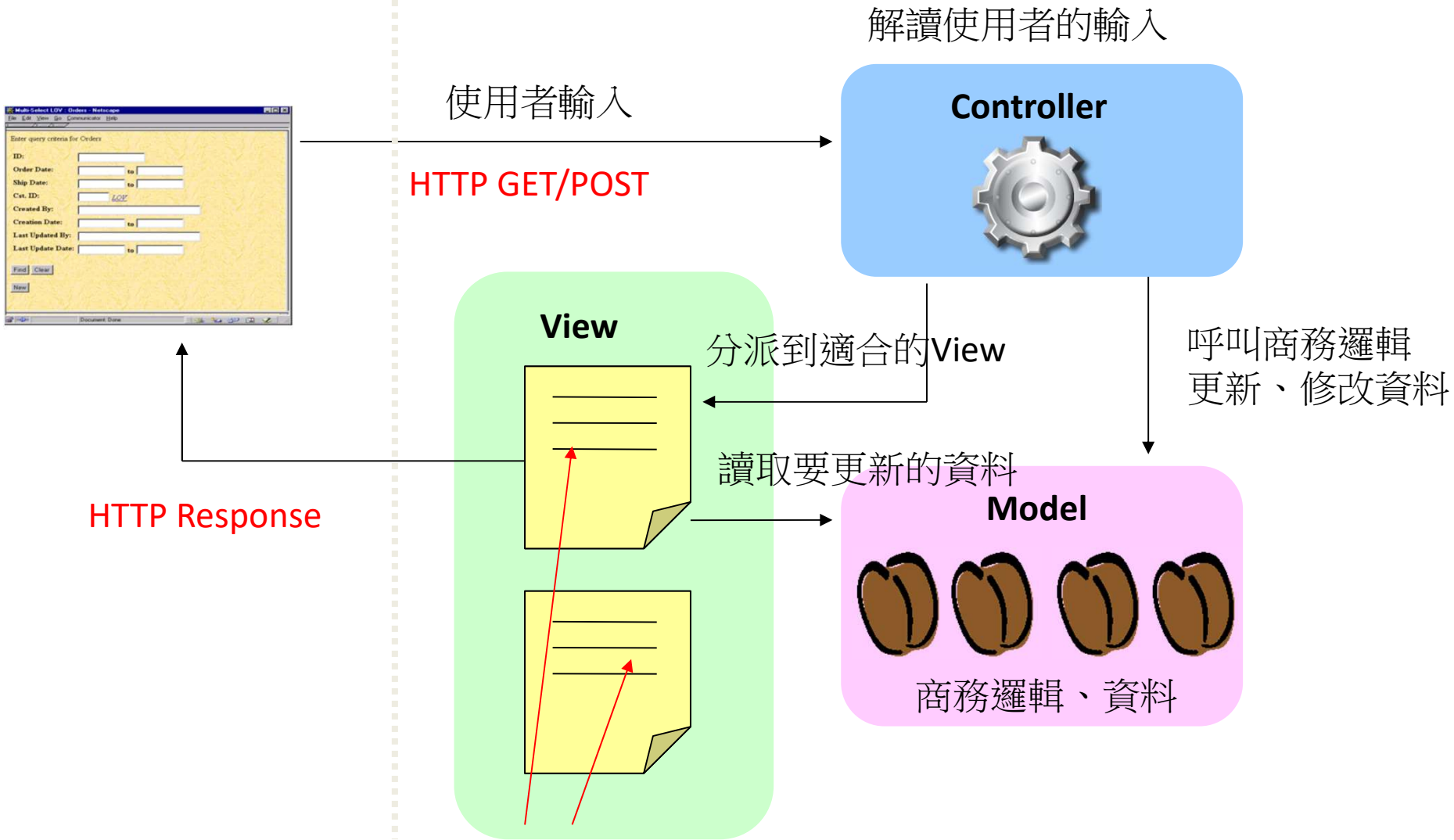
MVC (Desktop)


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

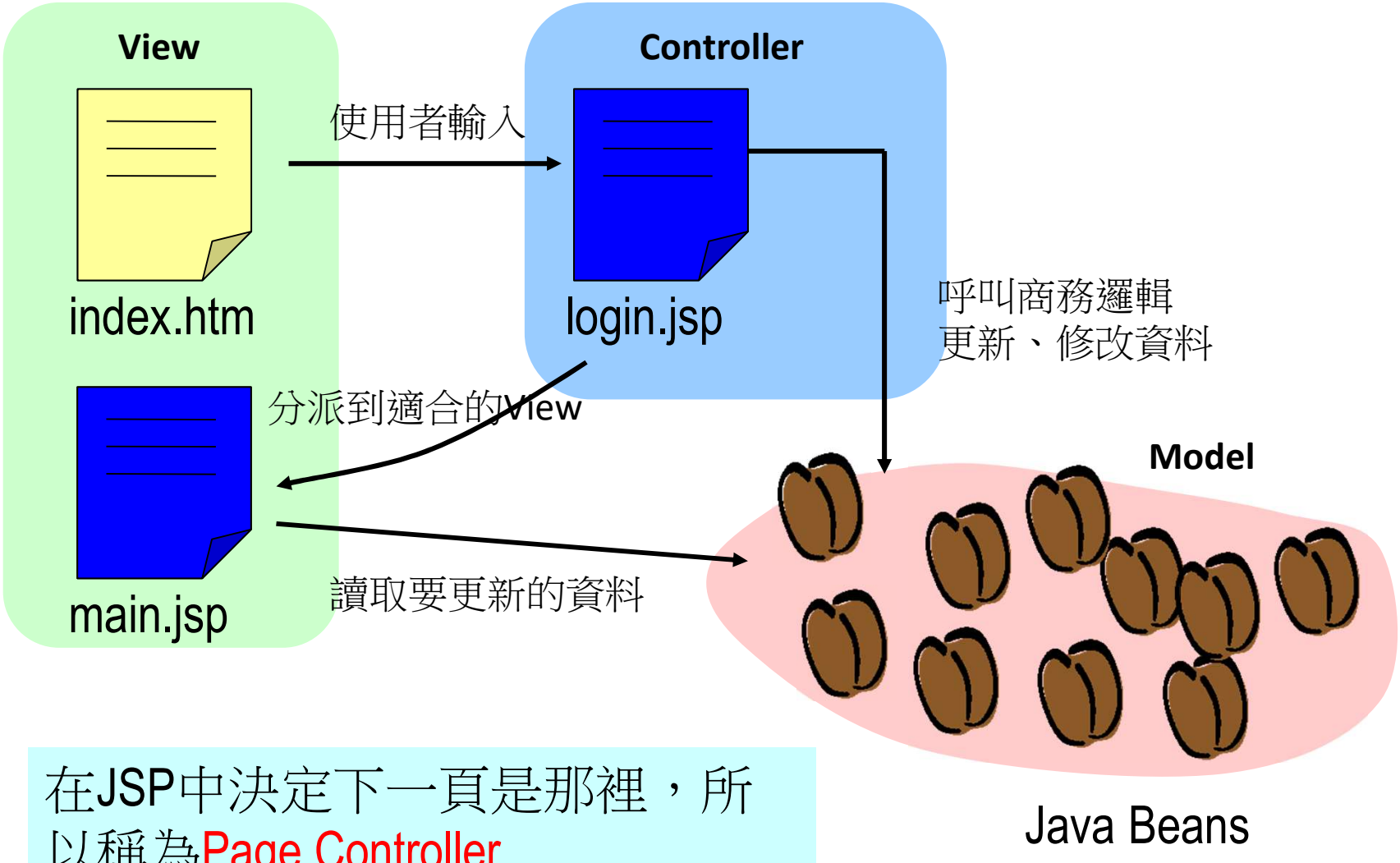


MVC (Web)



JSP被當成文件，先在server產生為靜態檔，再傳回client

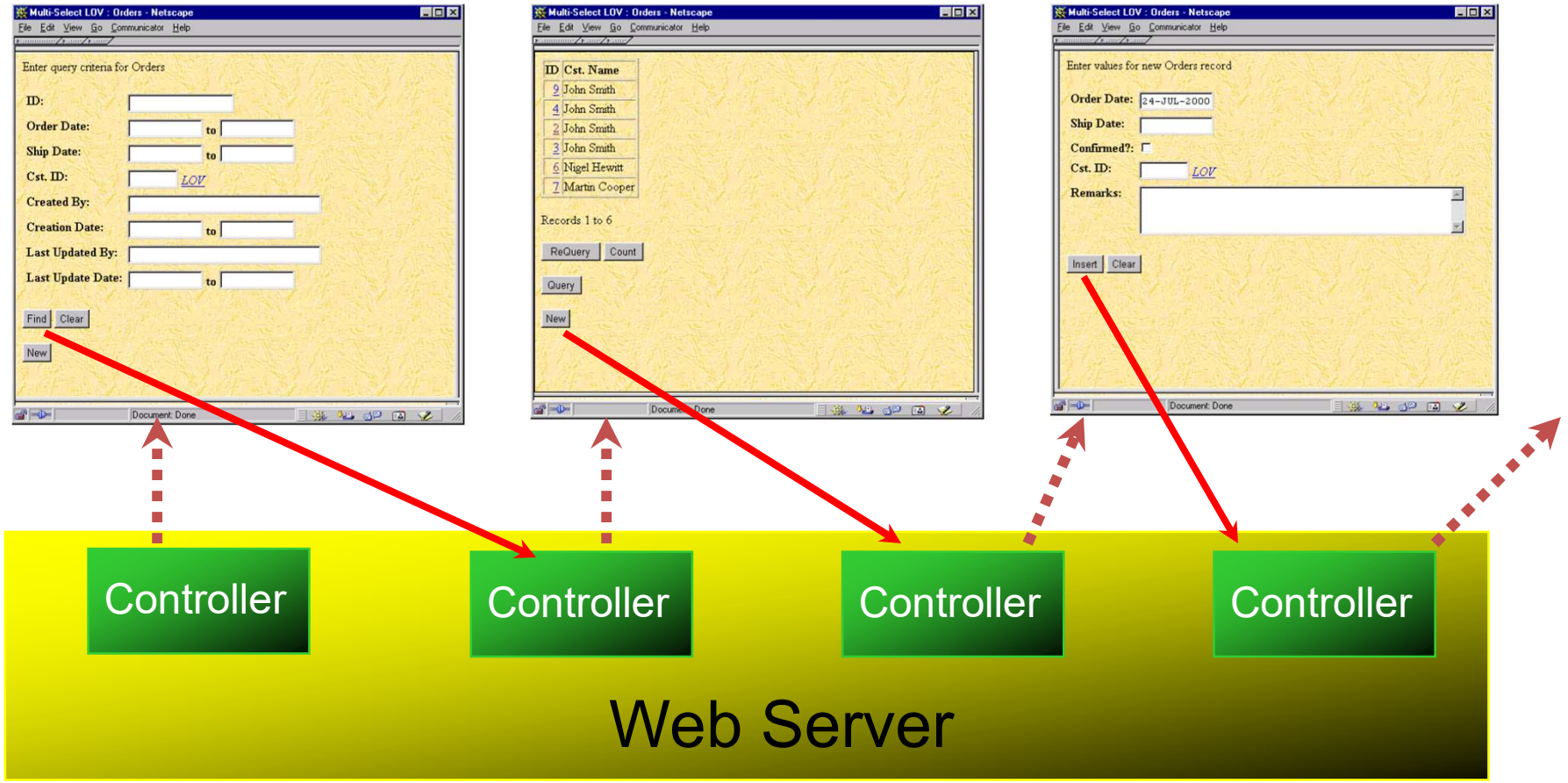
Model 1 : Page Controller



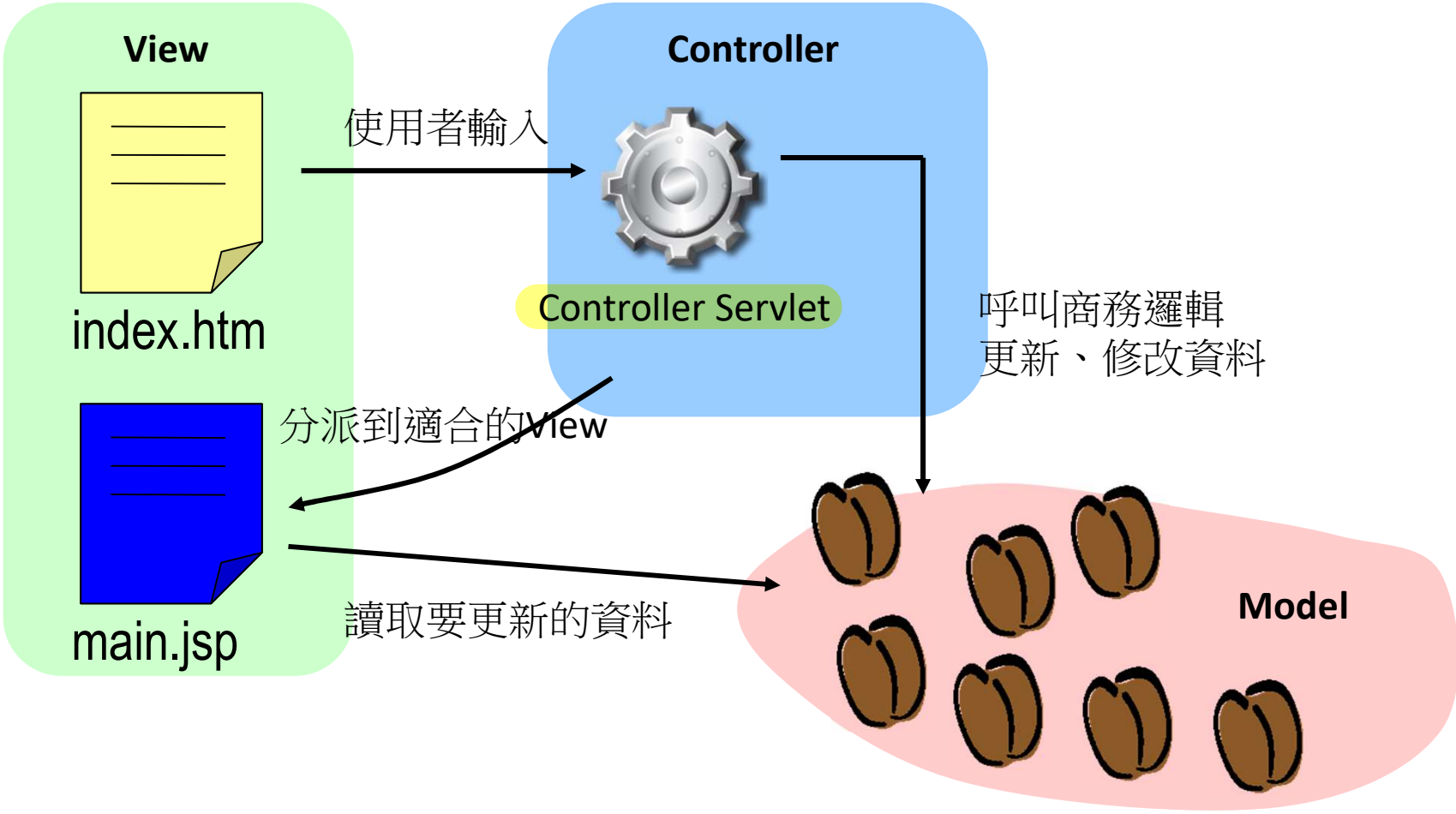
在JSP中決定下一頁是那裡，所以稱為Page Controller

Model 1有什麼問題？

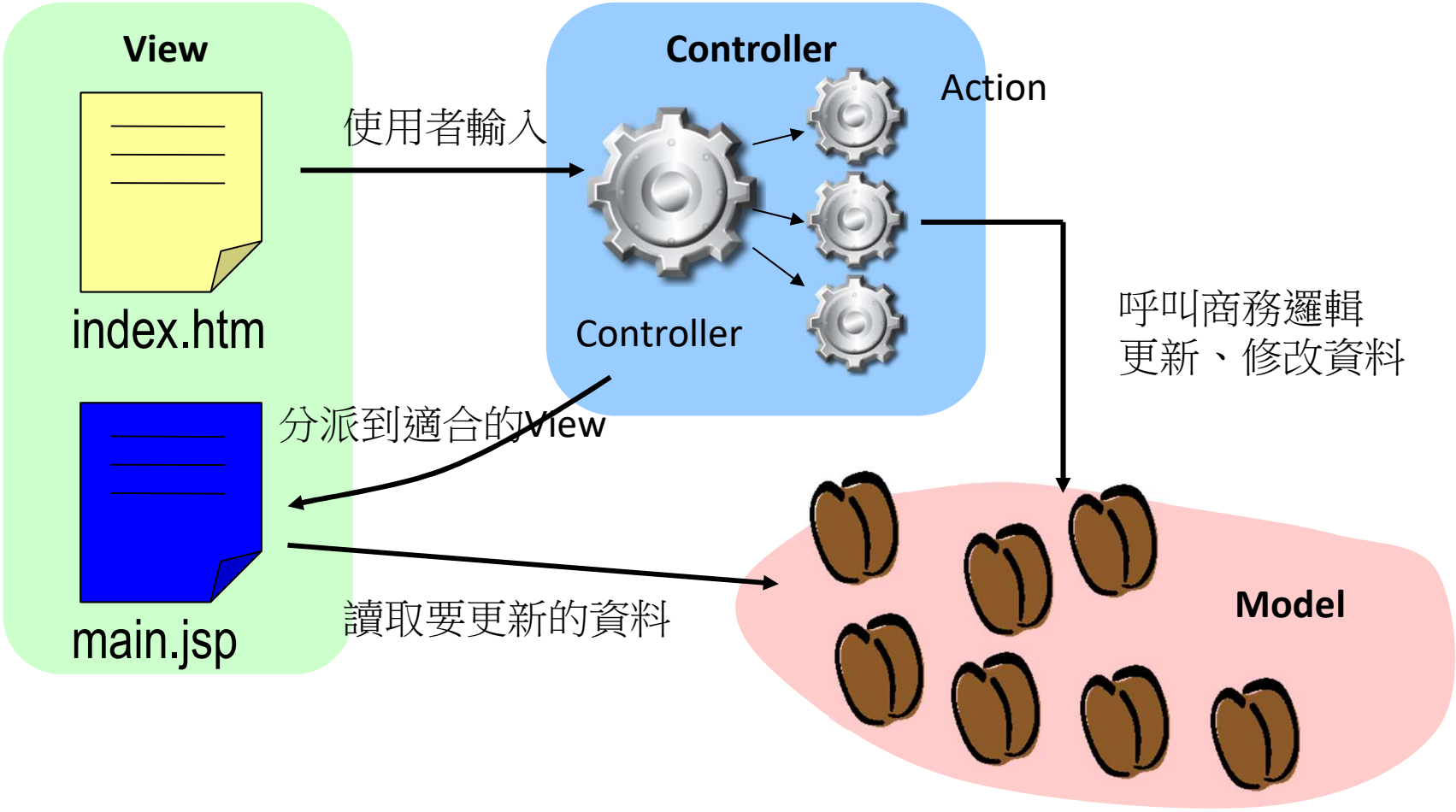
One controller per page , hard to maintain !



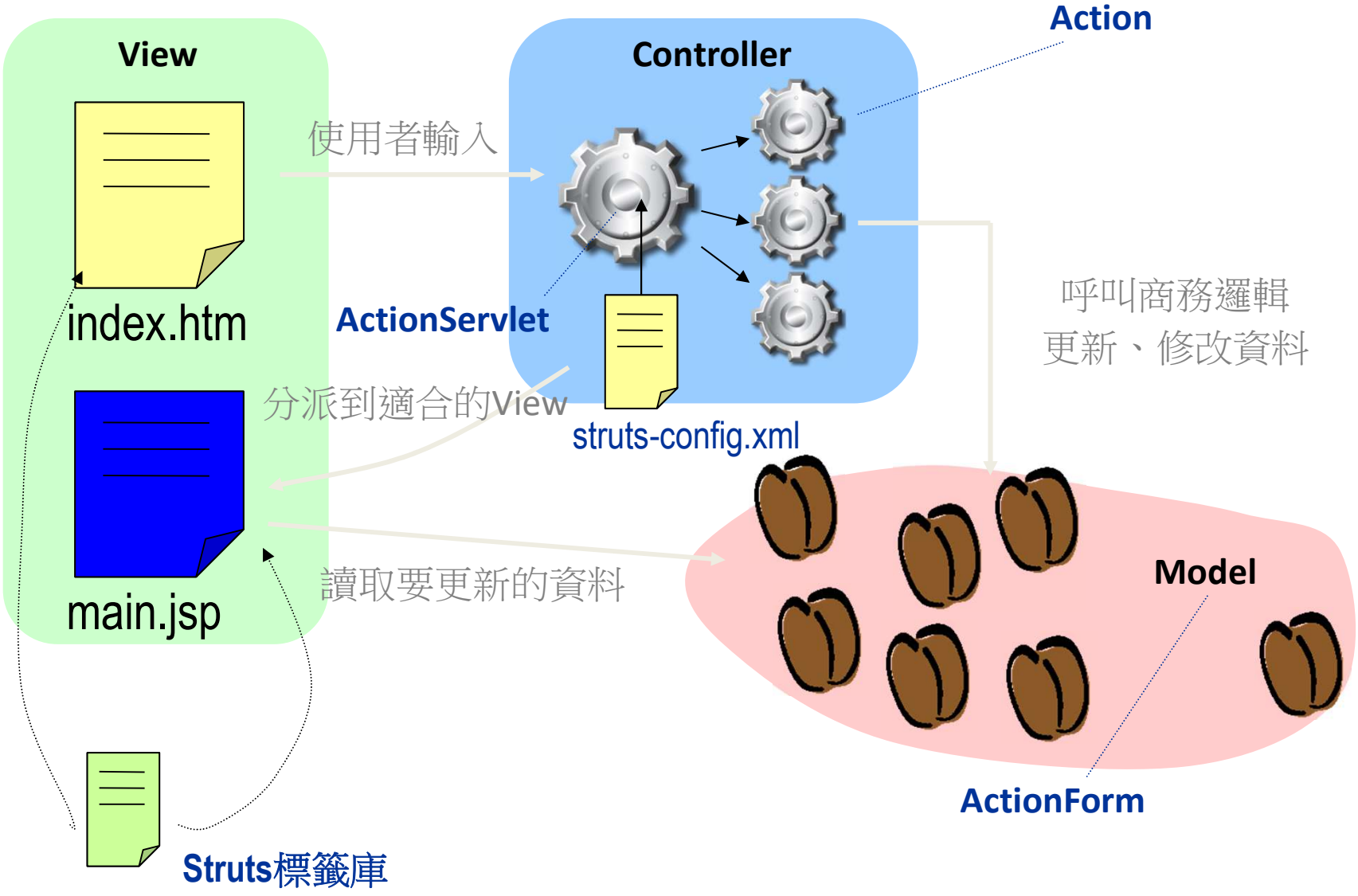
Model 2


View
Controller
使用者輸入
index.htm
Controller Servlet
呼叫商務邏輯
更新、修改資料
分派到適合的View
main.jsp
讀取要更新的資料
Model

將Controller模組化

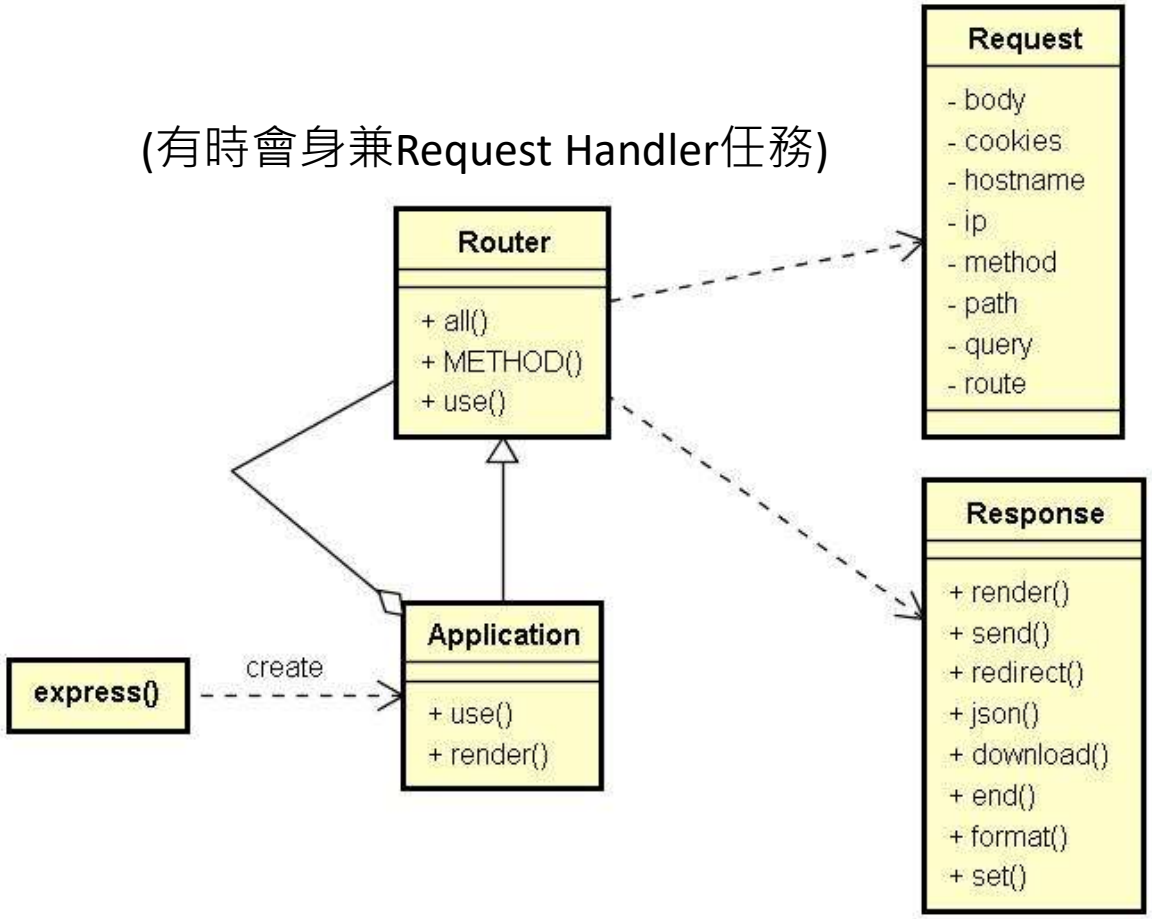


Struts Framework

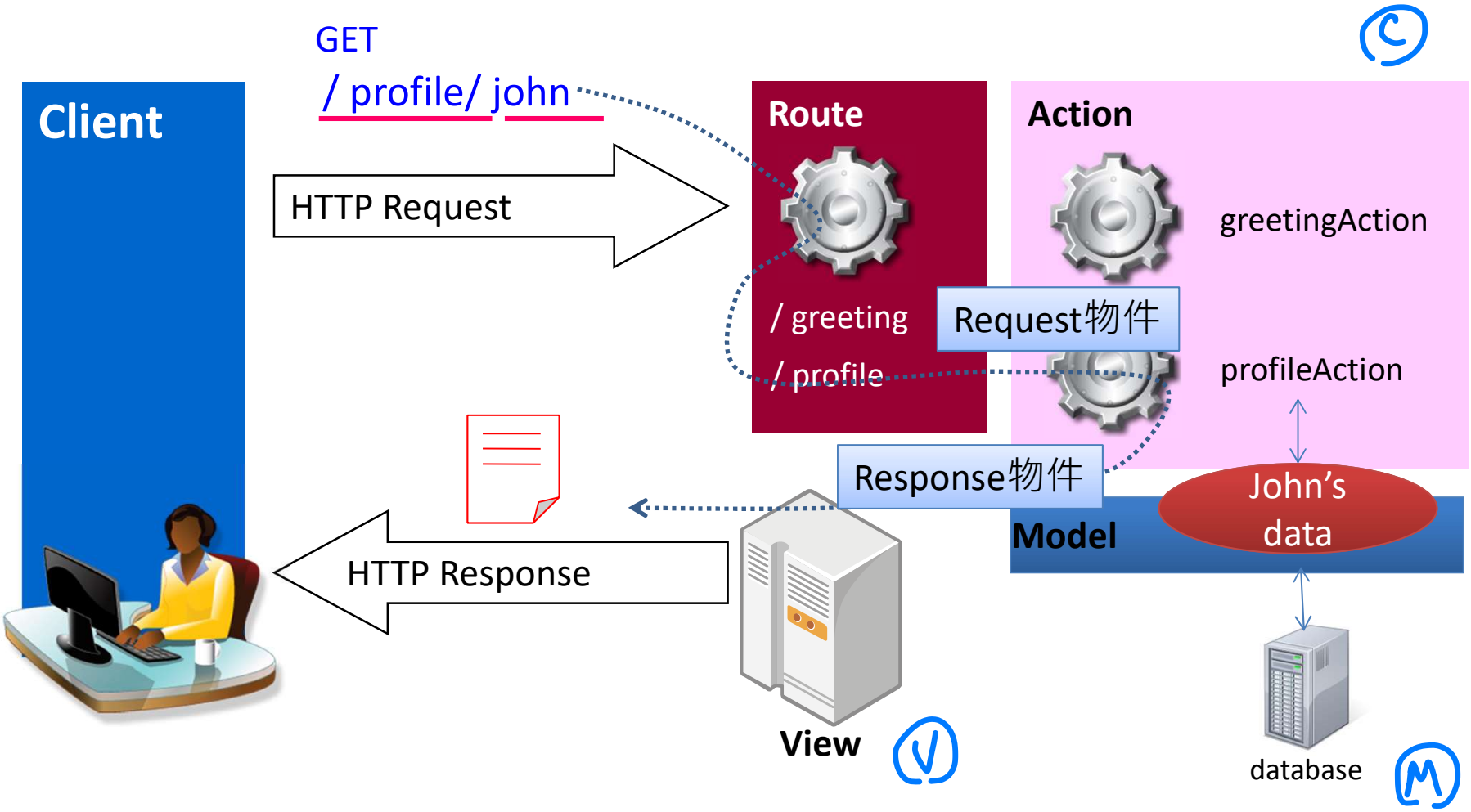


Express結構

(有時會身兼Request Handler任務)

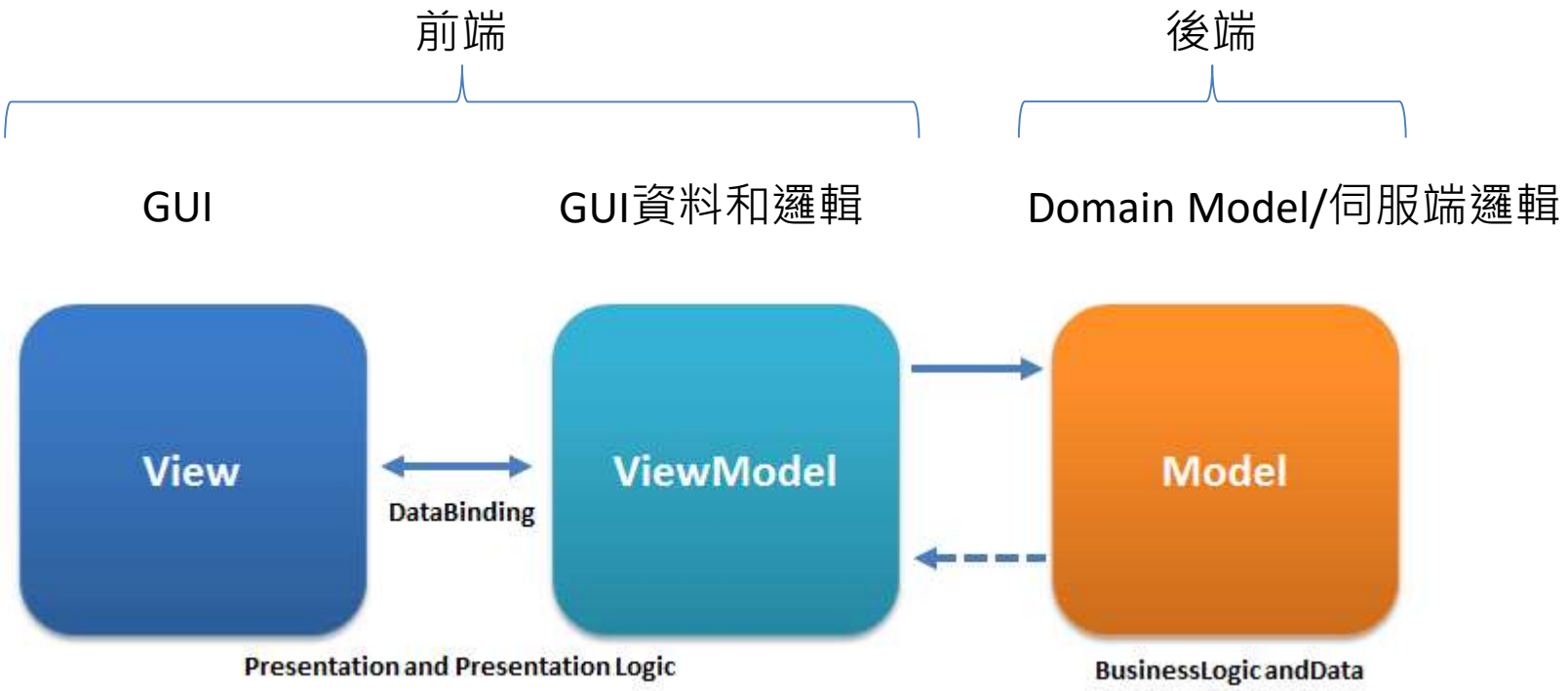


Express MVC處理架構



MVVM

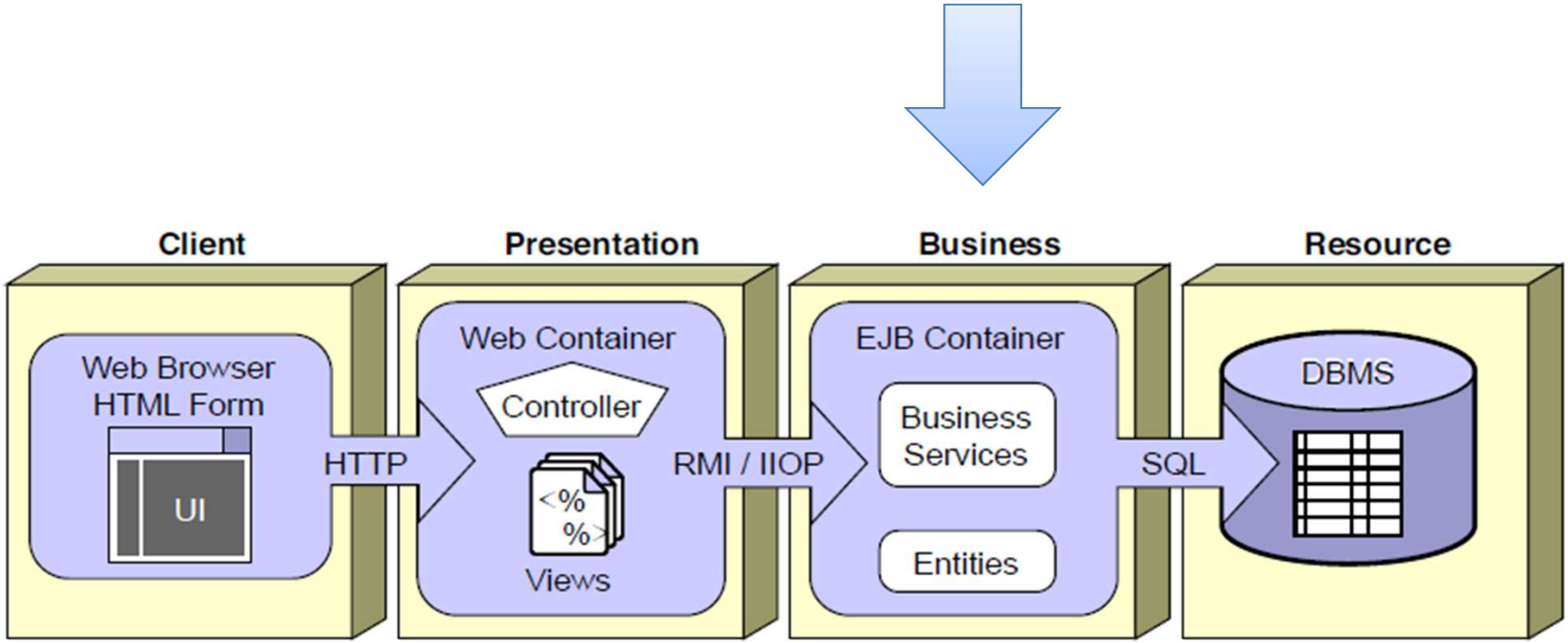
Introducing ViewModel and Binder
ViewModel 和 Model間資料不一定要即時同步，因此交通量會較MVC低
將大部份GUI 事務由後台切割出來 (後台角色由Controller→Web API)



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

中介服務架構


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

EJB Component Types

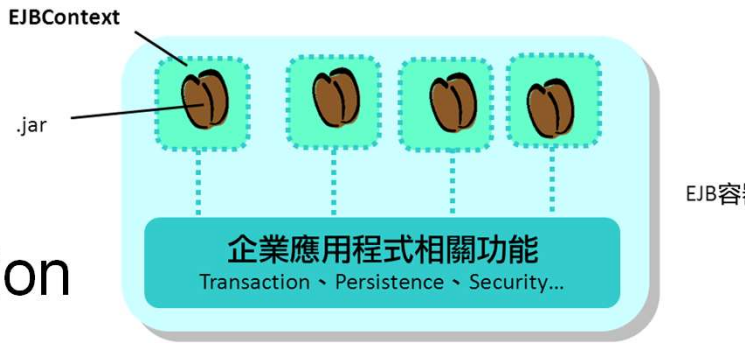
- Business logic
  - 可以想成remote function call
  - Session Beans (Stateless, stateful)
- Messaging
  - 可以想成remote event processor
  - Message Driven Beans
- Persistent entity
  - Entity Beans

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

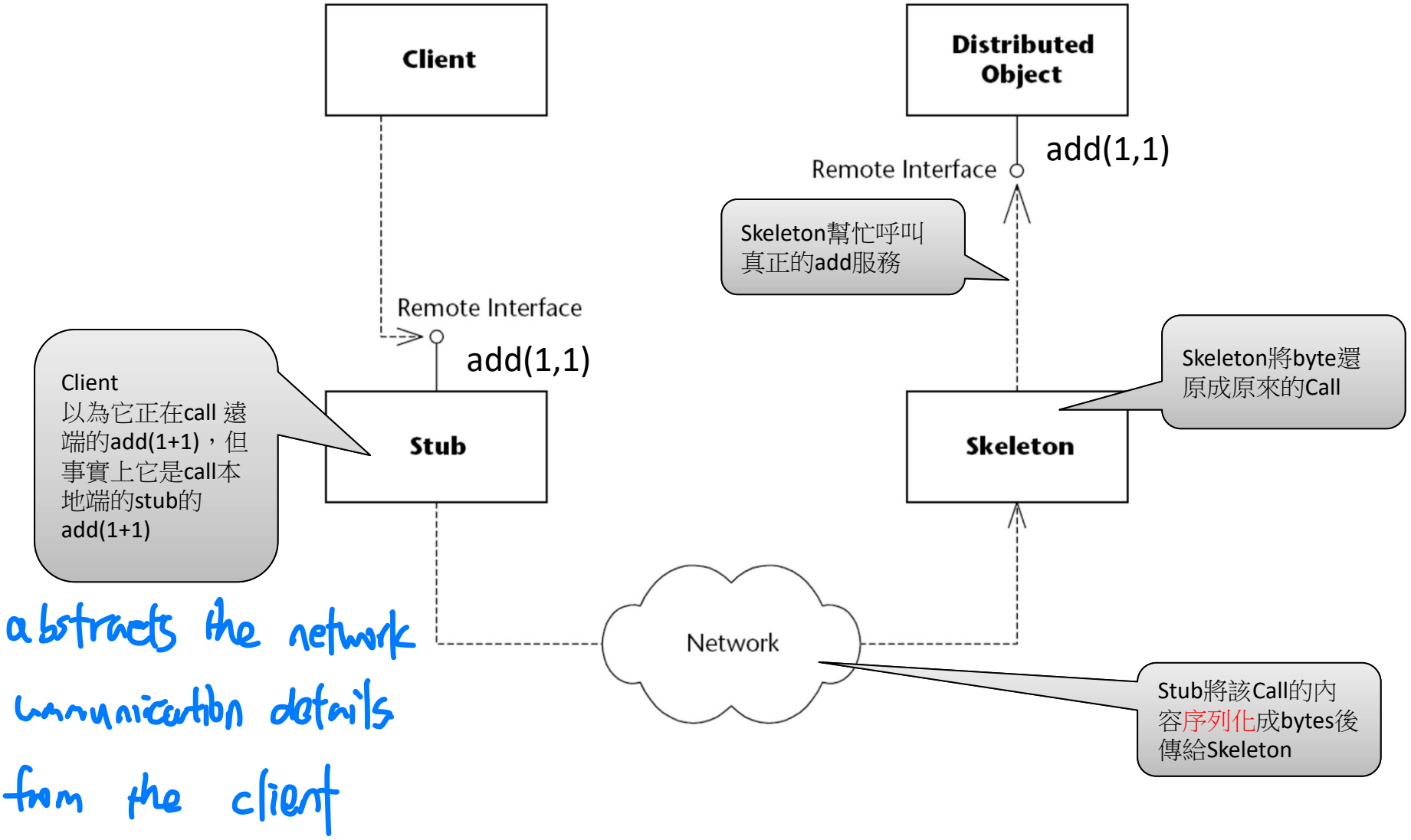
```
public  void
onMessage(Message message)
{
    ....

}
```

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

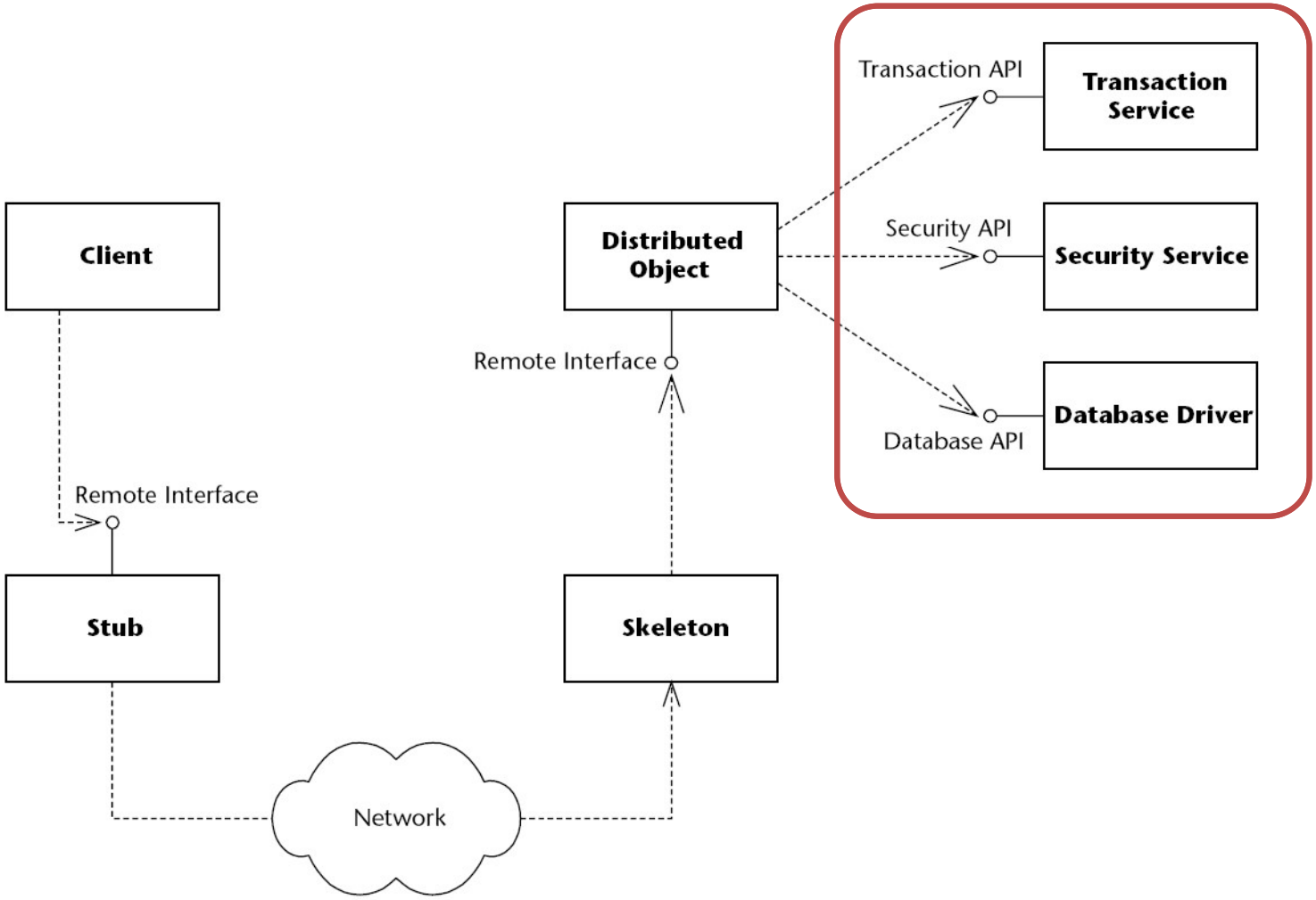


典型呼叫遠端物件的架構

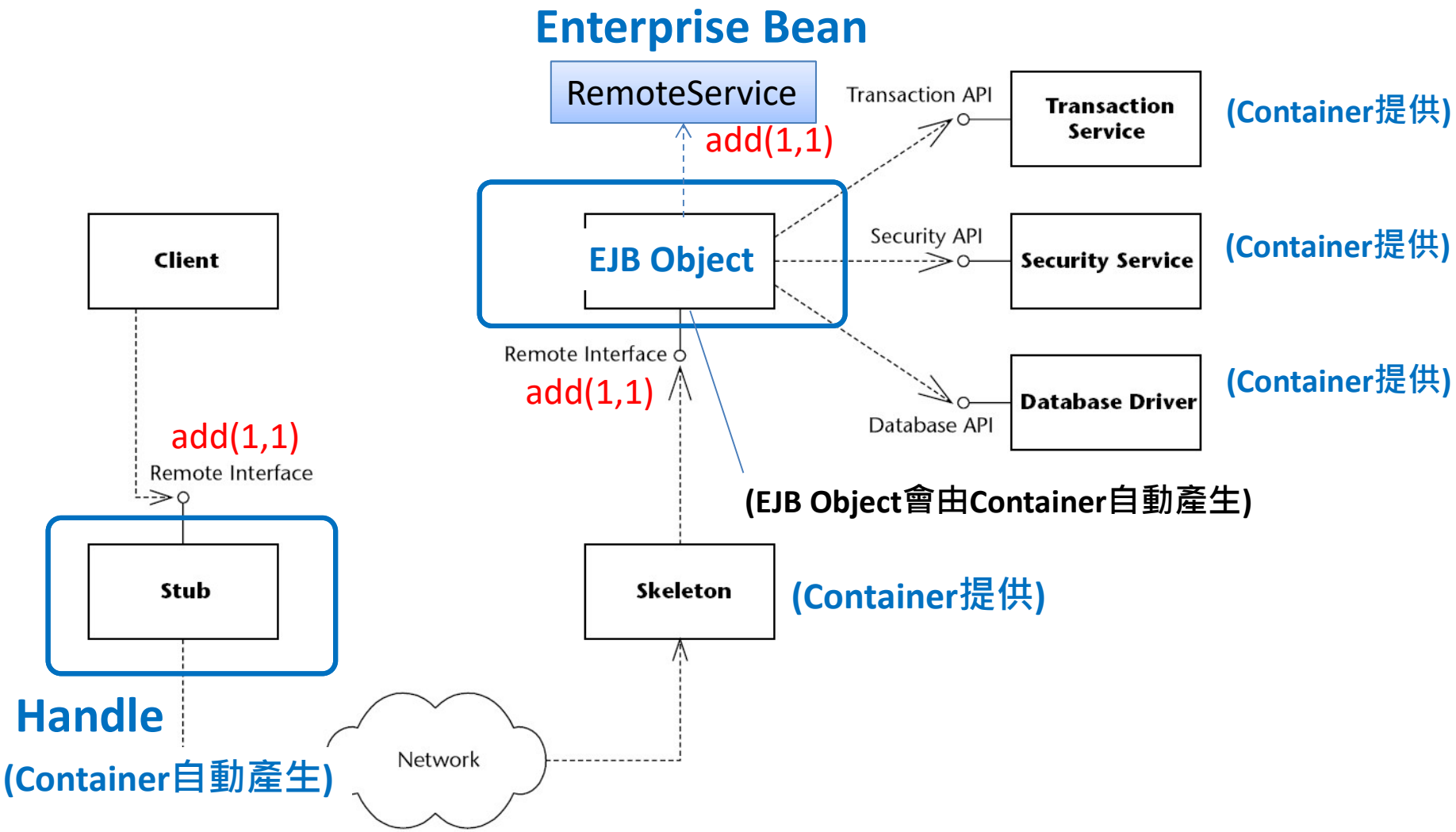


abstracts the network communication details from the client

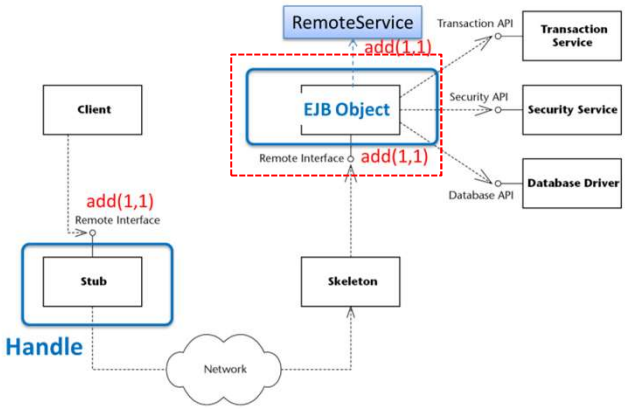
EJB容器在Skeleton中
順便幫我們加入了其它服務...



在EJB容器中我們這樣稱呼它們

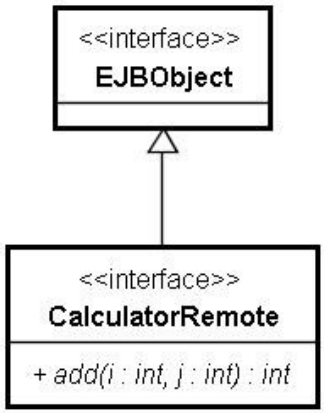


產生EJB Object



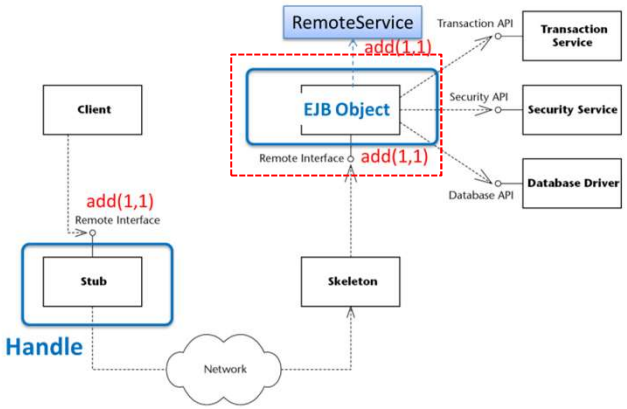
首先，要宣告遠端服務的介面與方法，例如 public int add(int i, int j)
這個介面稱為Remote Interface

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



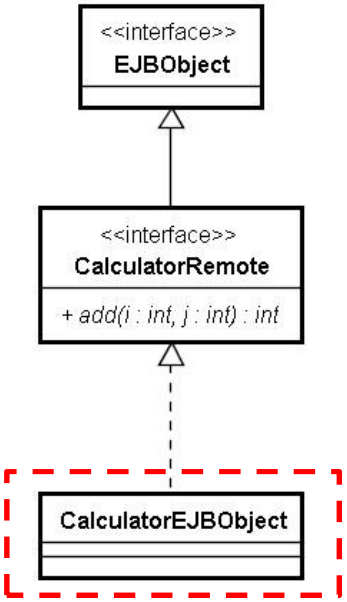

```
public interface CalculatorRemote extends EJBObject {

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

}
```

產生EJB Object



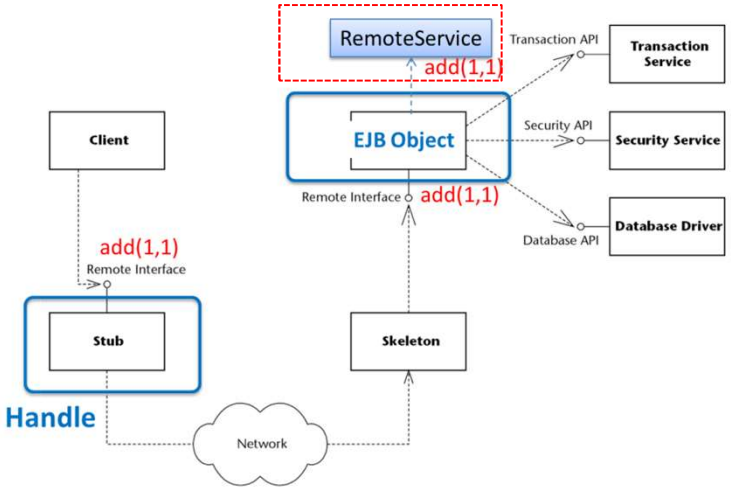
依照你宣告的interface內容，

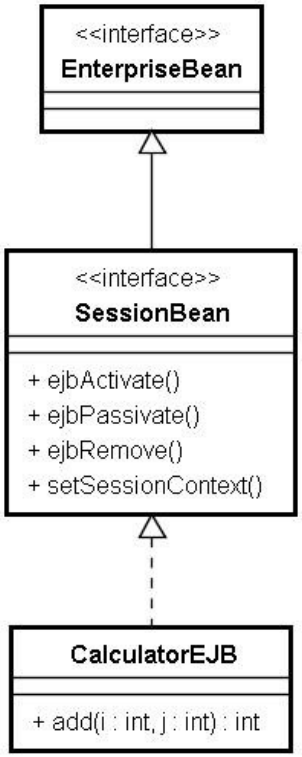
EJB容器會自動幫你產生專屬的EJBObject



自動產生

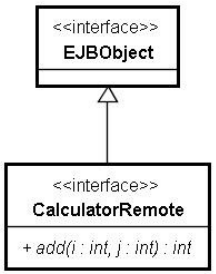
寫真正的服務





真正的服務必須要實作任意一個EnterpriseBean的子介面

例如”SessionBean”




```
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);
```

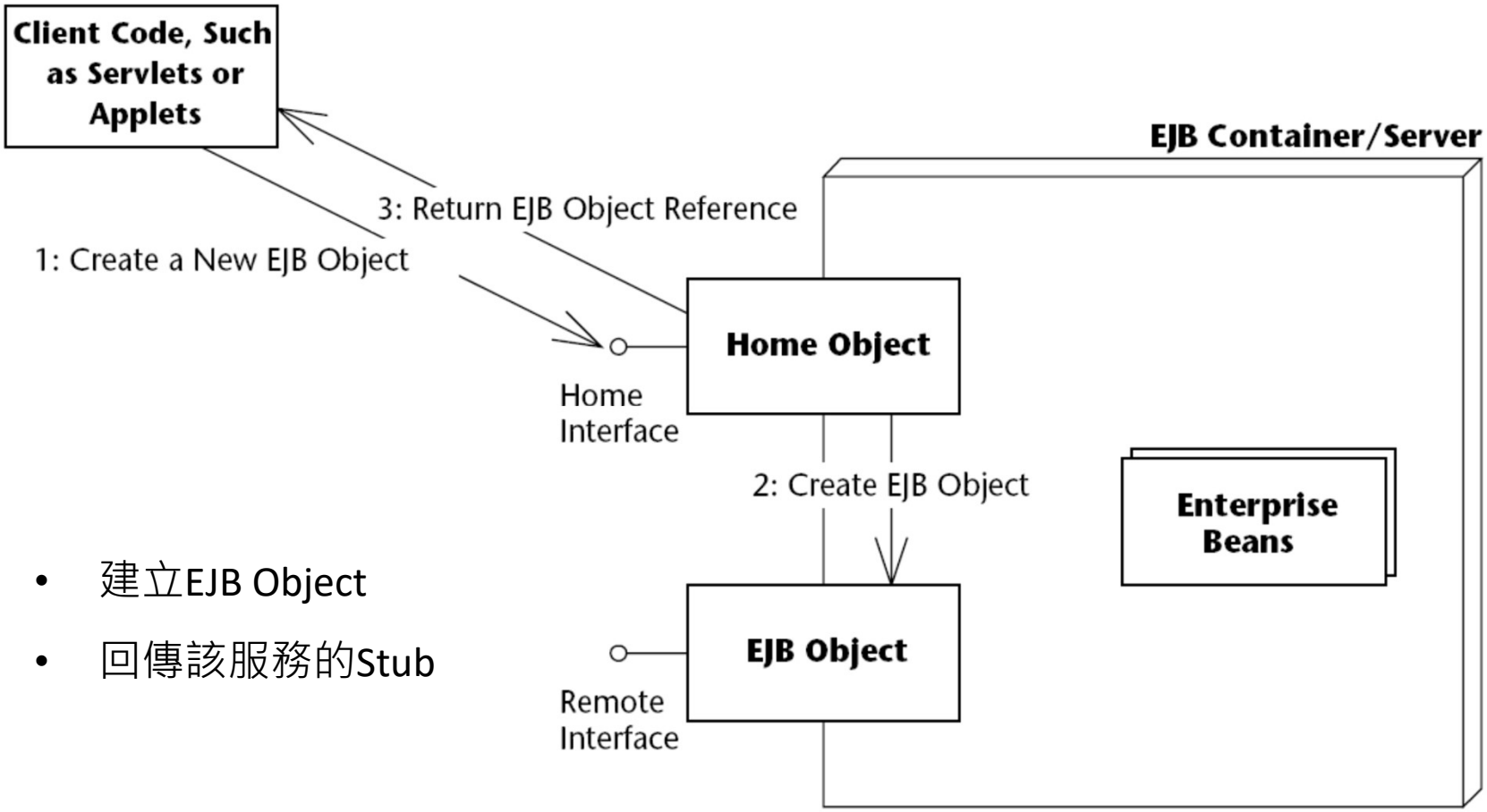
- 如何new遠端的物件?
  - 需要使用EJB Home

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

EJB Home何處尋?

EJB Home

EJB Home的主要功能

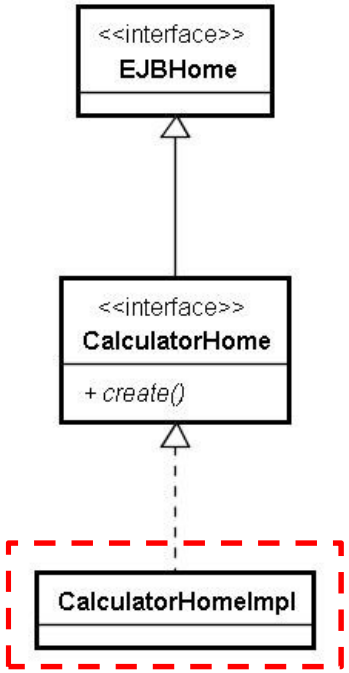


- 建立EJB Object
- 回傳該服務的Stub

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 (掛在JNDI上)

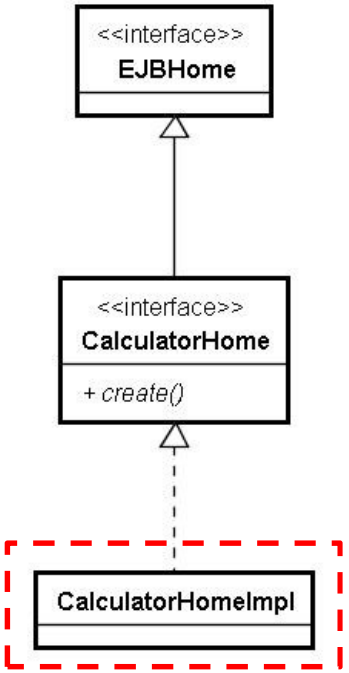


自動產生

Java Naming and Directory Interface

CalculatorHome

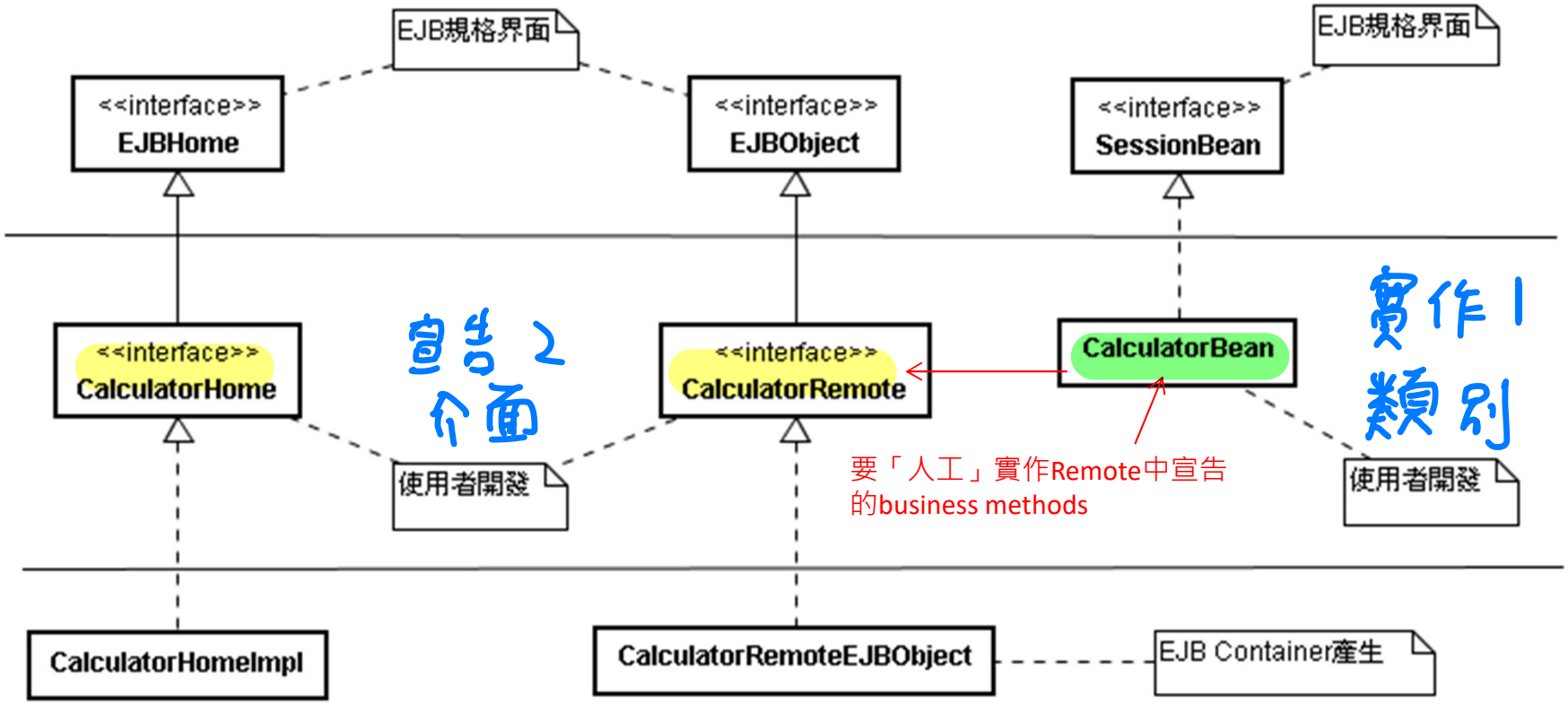
```
public interface CalculatorHome extends EJBHome {

    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



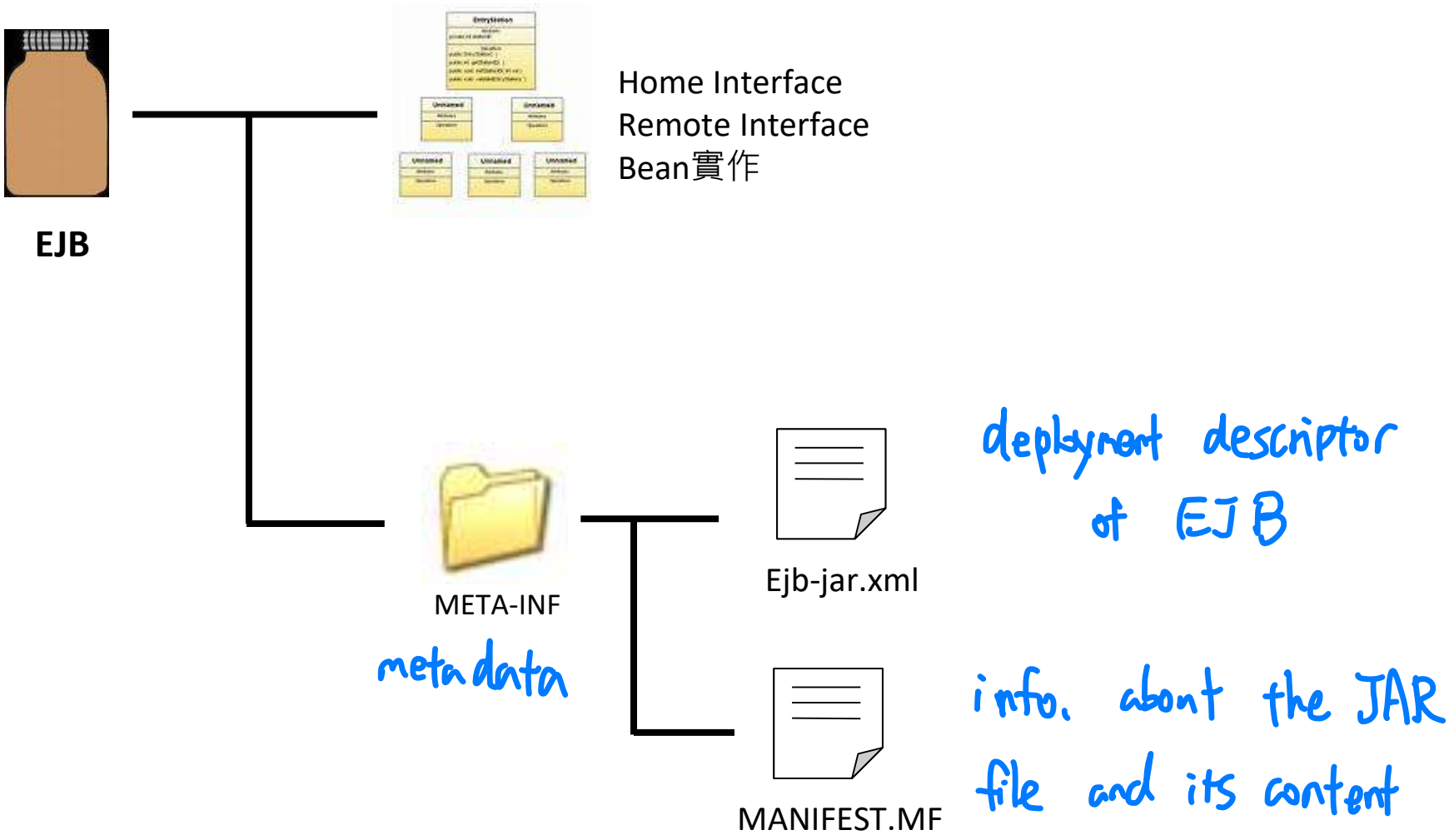
如何手工打造EJB元件

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)

JAR : JAVA Archive

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
metadata
Ejb-jar.xml
deployment descriptor of EJB
MANIFEST.MF
info. about the JAR file and its content

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("CalculatorEJB");
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);
```

這裡拿到的是遠端元件的Stub，並非實體

CalculatorRemote calculator;

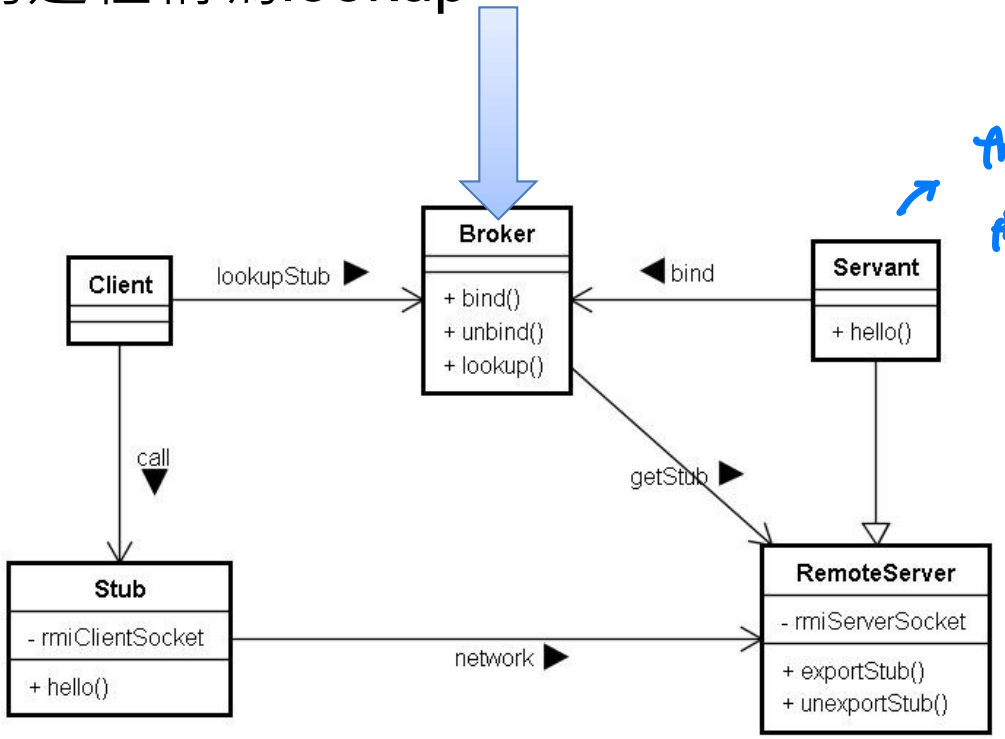
(Dependency Injection)

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

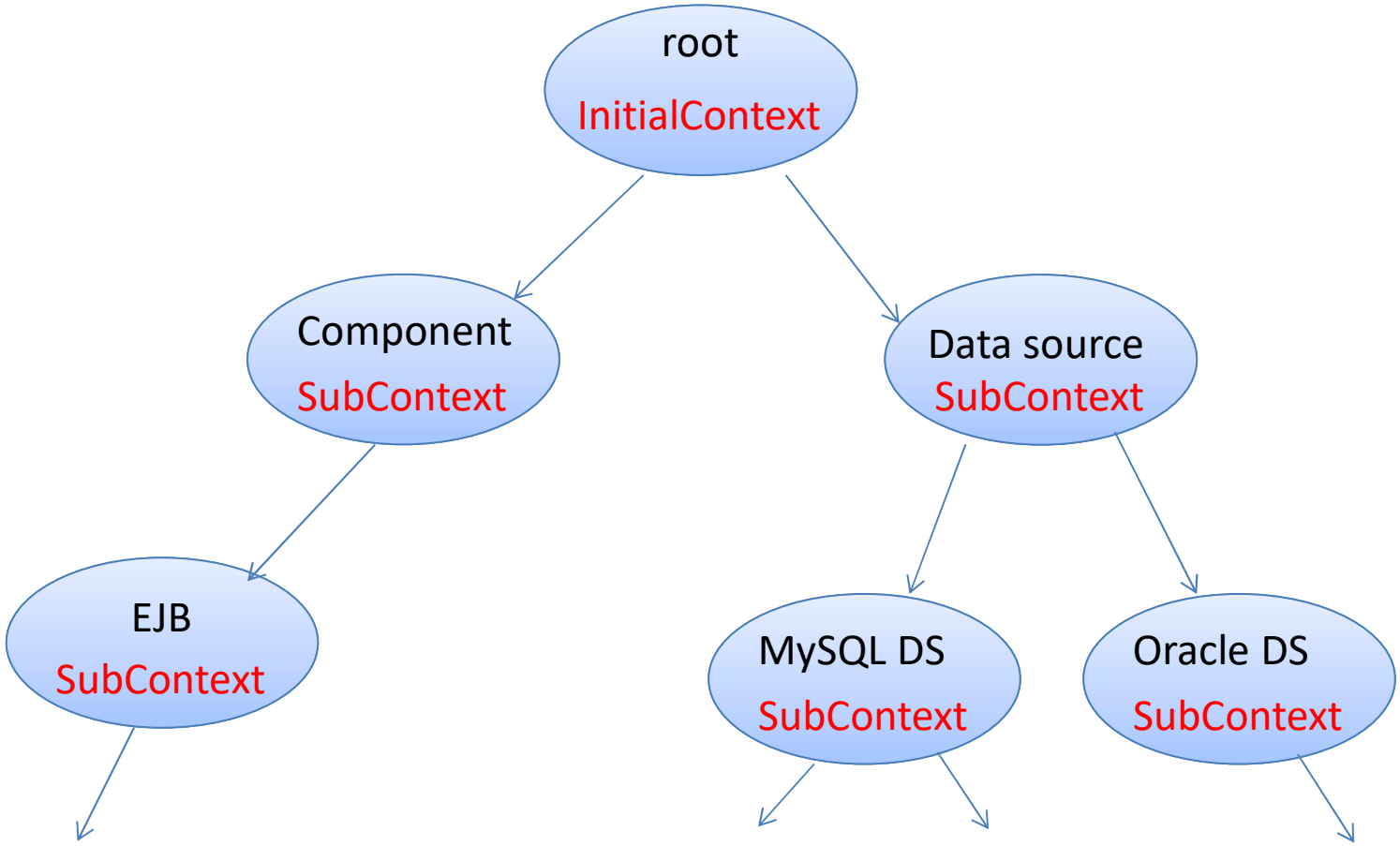
Java Naming and Directory Interface (JNDI)

- 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

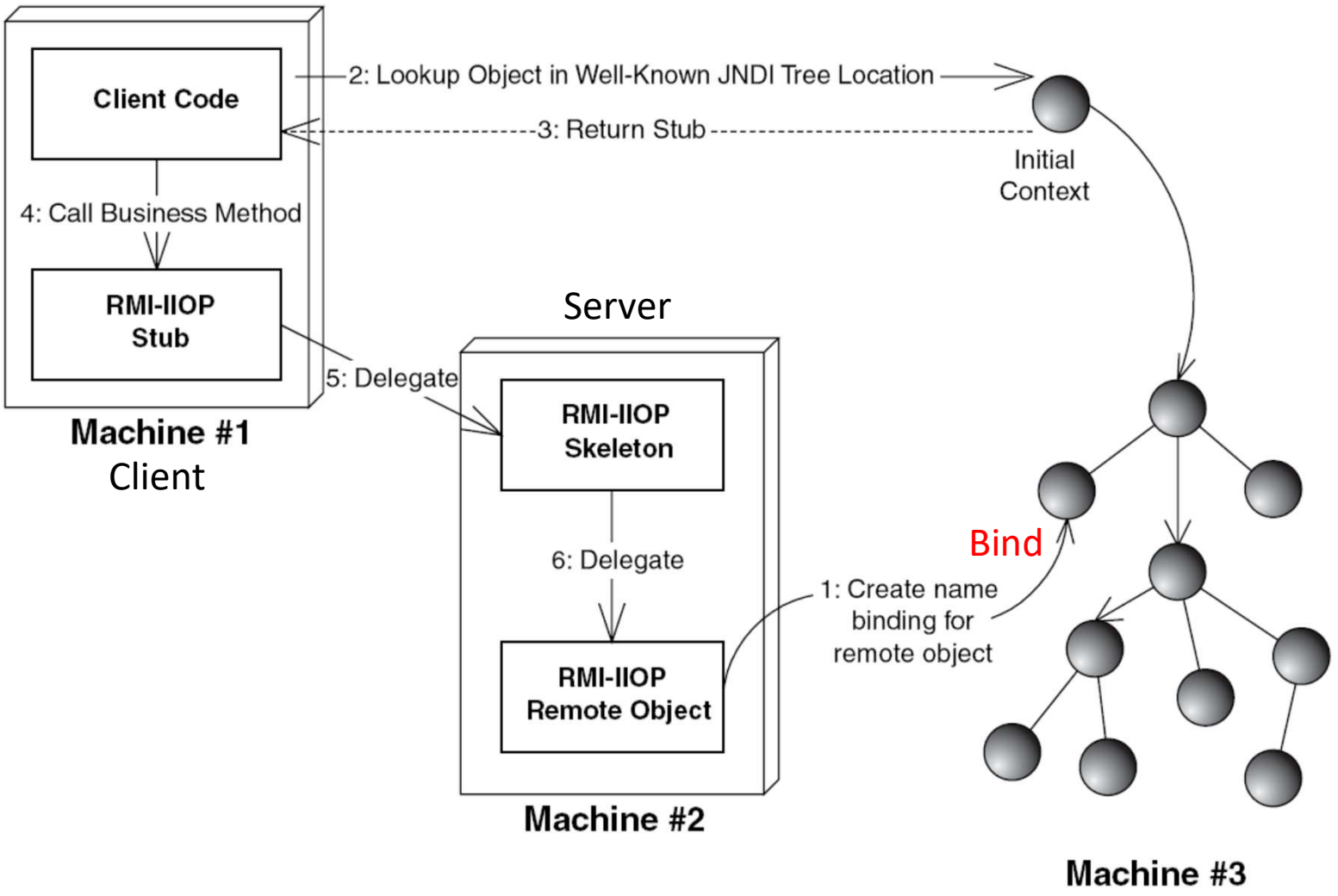


the server-side object that implements the actual service
e.g. EJB

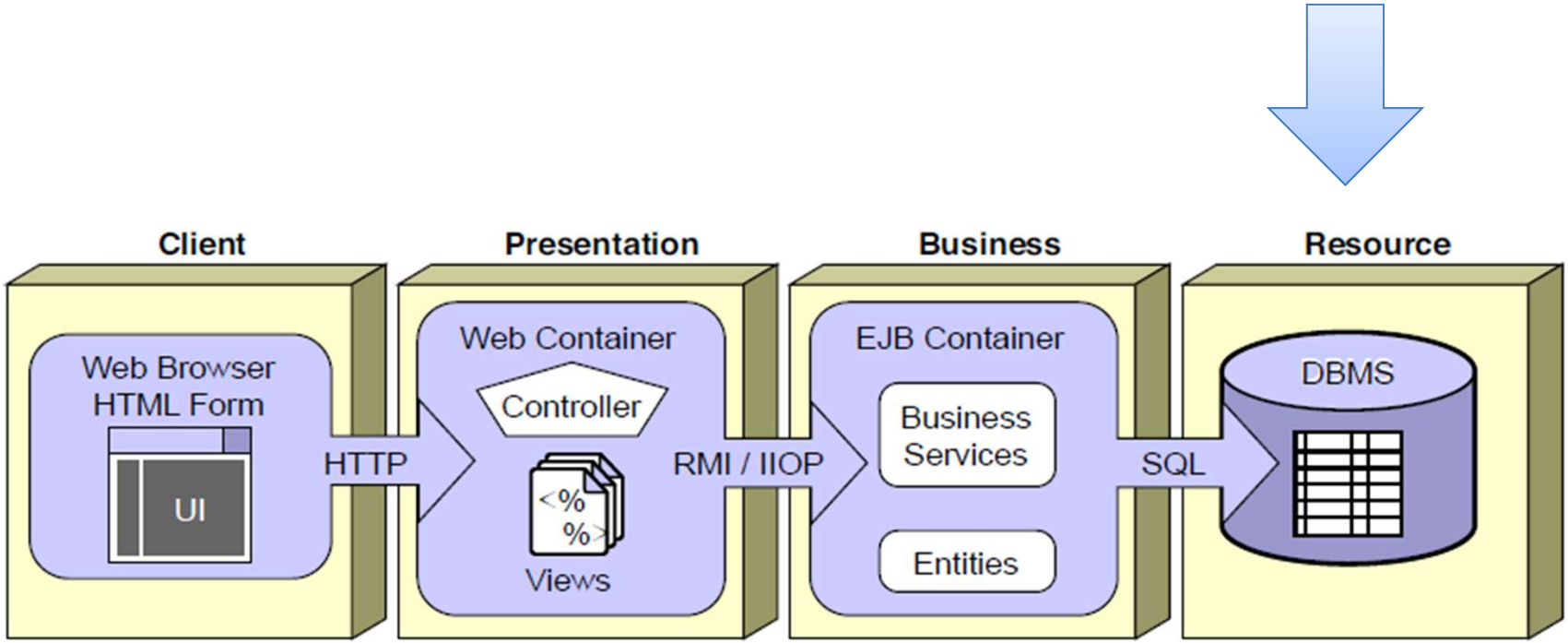
JNDI Tree


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

JNDI運作架構


Client Code
2: Lookup Object in Well-Known JNDI Tree Location
3: Return Stub
Initial Context
4: Call Business Method
RMI-IIOP Stub
Server
5: Delegate
RMI-IIOP Skeleton
6: Delegate
RMI-IIOP Remote Object
Bind
1: Create name binding for remote object
Machine #1
Client
Machine #2
Machine #3

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

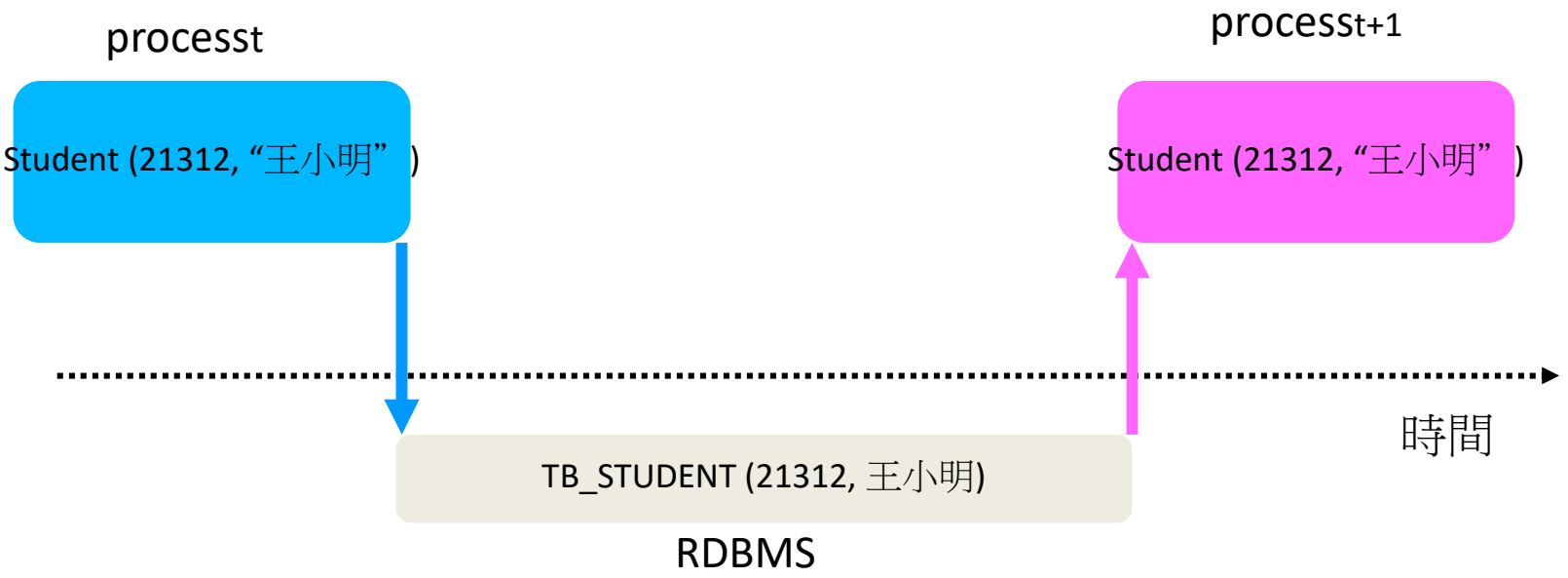
物件的永續性 (Persistence)

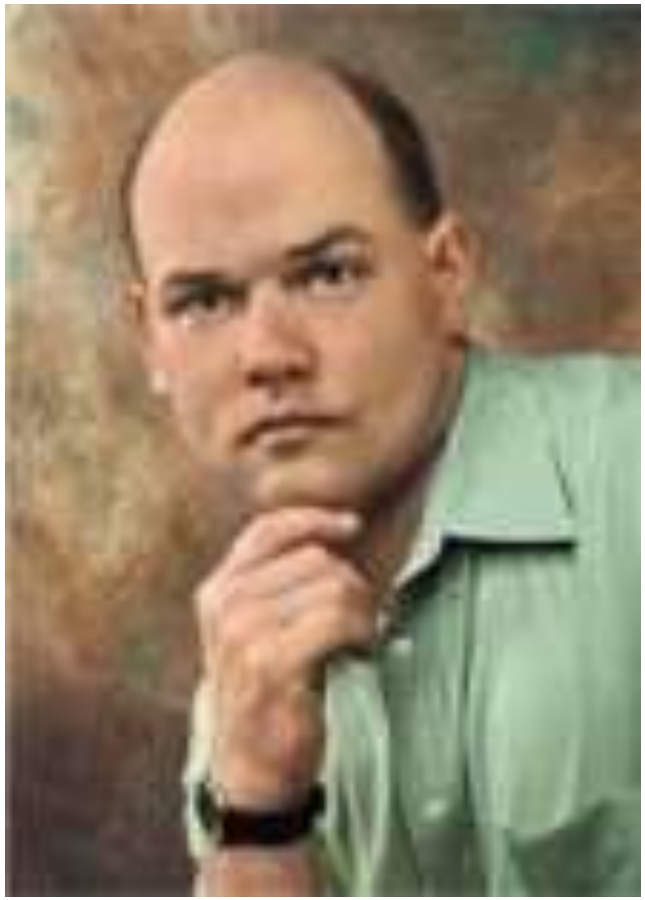
- Java應用程式中的物件存在記憶體(Heap)中

→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





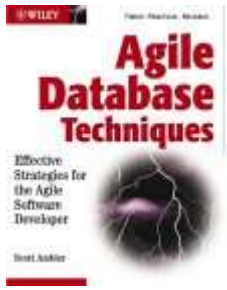
S.W. Ambler

Java的永續性解決方案

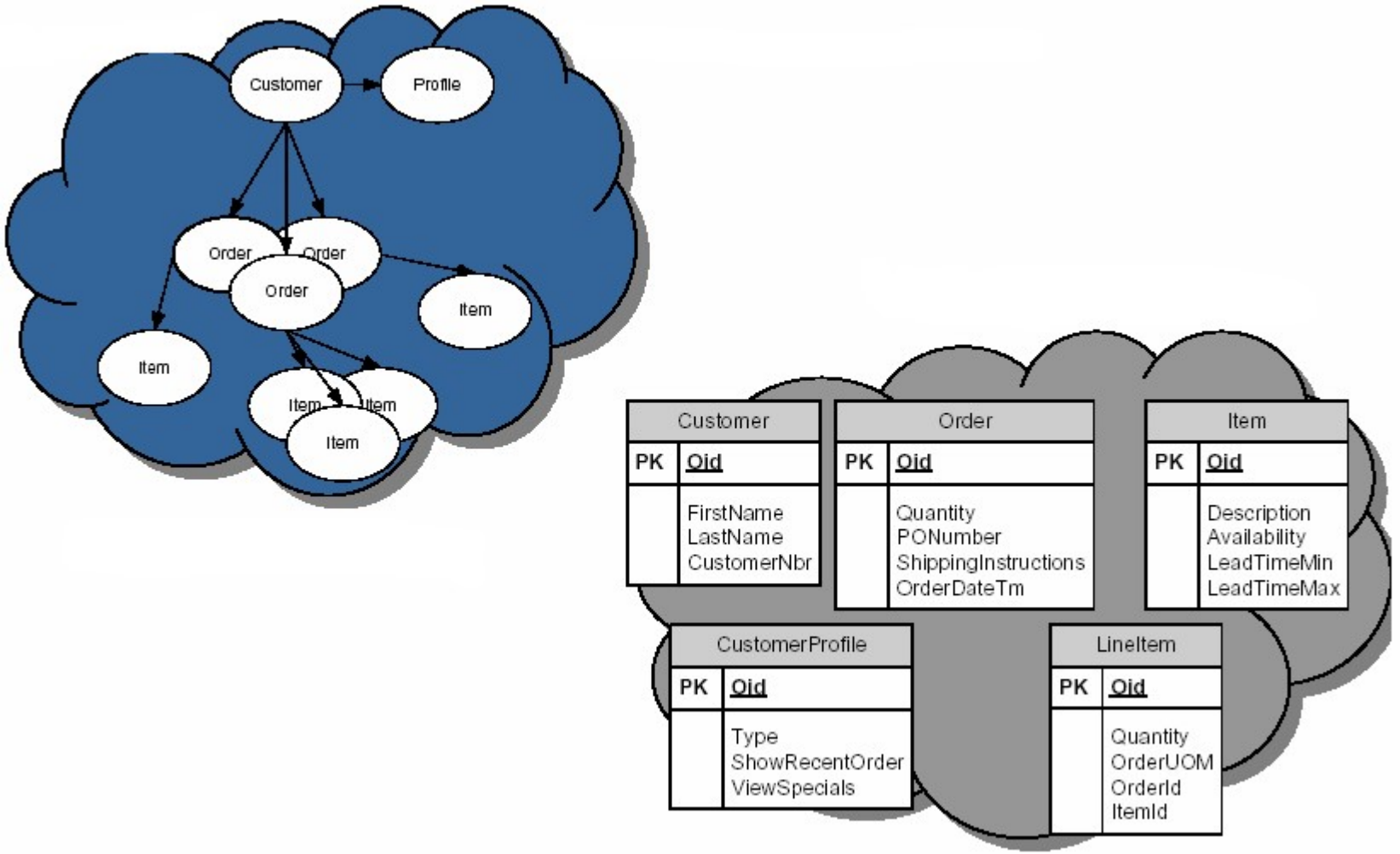
- 關聯式資料庫系統(RDBMS)
  - 企業應用程式最常使用的永續性機制

- 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


Customer
Profile
Order
Order
Order
Item
Item
Item
Item
Item
Customer
PK
Oid
FirstName
LastName
CustomerNbr
Order
PK
Oid
Quantity
PONumber
ShippingInstructions
OrderDateTm
Item
PK
Oid
Description
Availability
LeadTimeMin
LeadTimeMax
CustomerProfile
PK
Oid
Type
ShowRecentOrder
ViewSpecials
LineItem
PK
Oid
Quantity
OrderUOM
OrderId
ItemId

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 (direct JDBC usage)
  - DAO (封裝JDBC的資料存取程式碼) Data Access Object
  - Apache DBUtils (JDBC Utility Component)
  - iBATIS SQLMap (現為Apache iBATIS)
  - 自己封裝 (custom wrappers)
  - Entity Beans (失敗)

Java永續性解決方案的演進 (續)

- 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)

- 標準Persistence Layer時期 (2006-)
  - Java Persistence API
- Java Persistence API (JPA)
  - POJO-based (Plain Old Java Objects)
    - 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

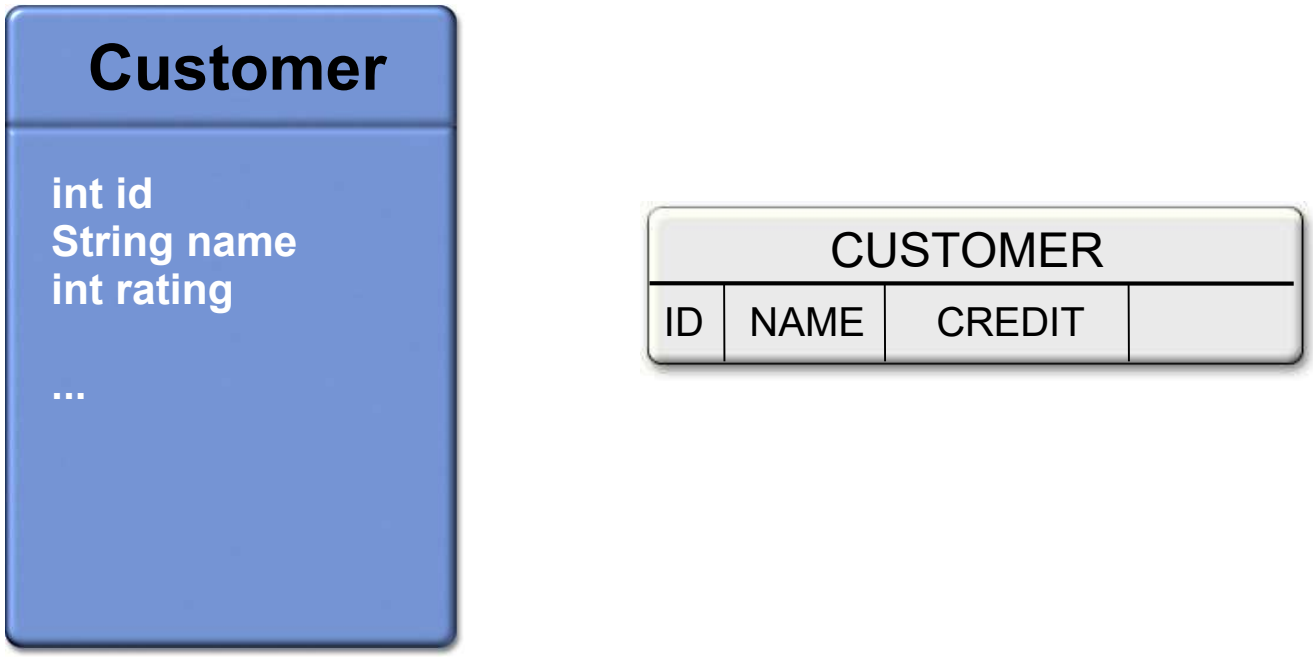
使用JPA的步驟

- 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

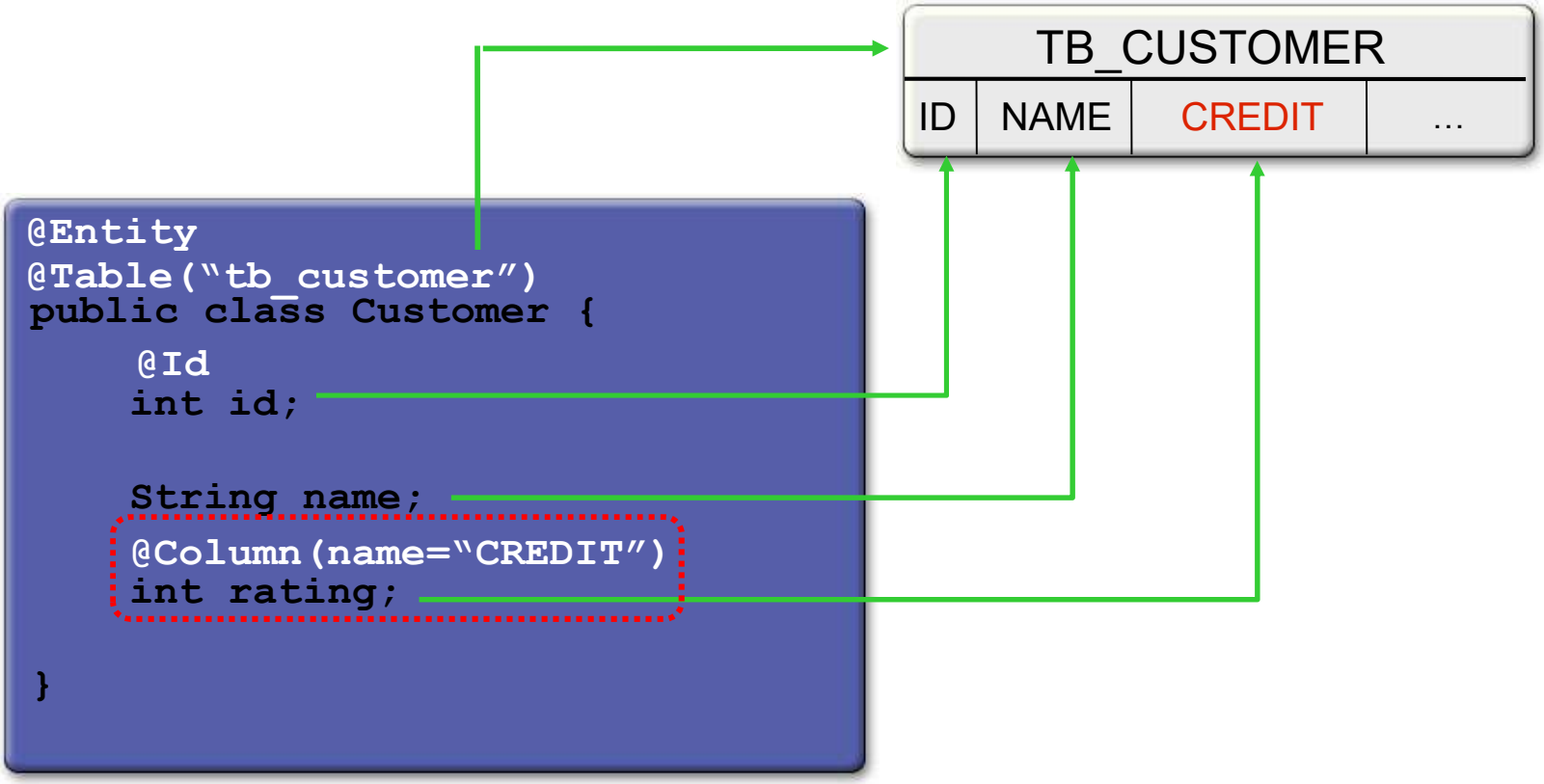
Step1 準備Library

- 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

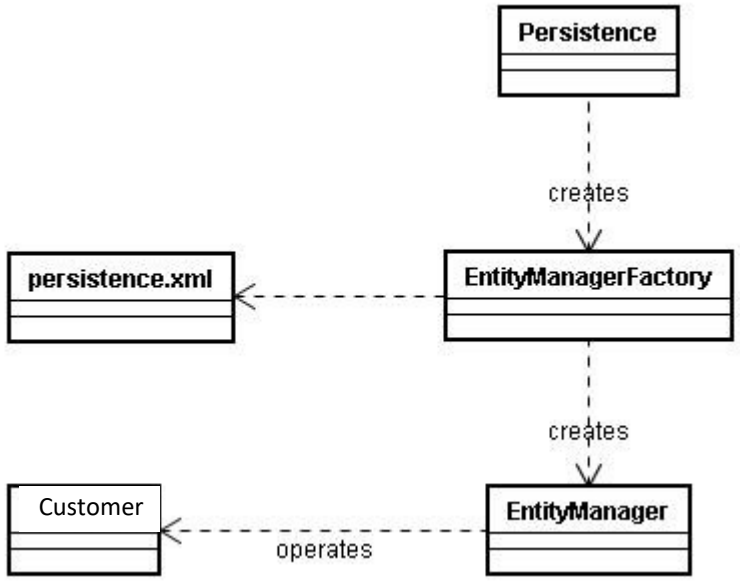
```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    version="1.0">
  <persistence-unit name="CustomerService">
    <class>Customer</class>
    <properties>
      <property name="toplink.jdbc.driver" value=" JDBC驅動程式全名"/>
      <property name="toplink.jdbc.url" value="資料庫的url"/>
      <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>
```

使用類別全稱

依使用的實作品有所不同

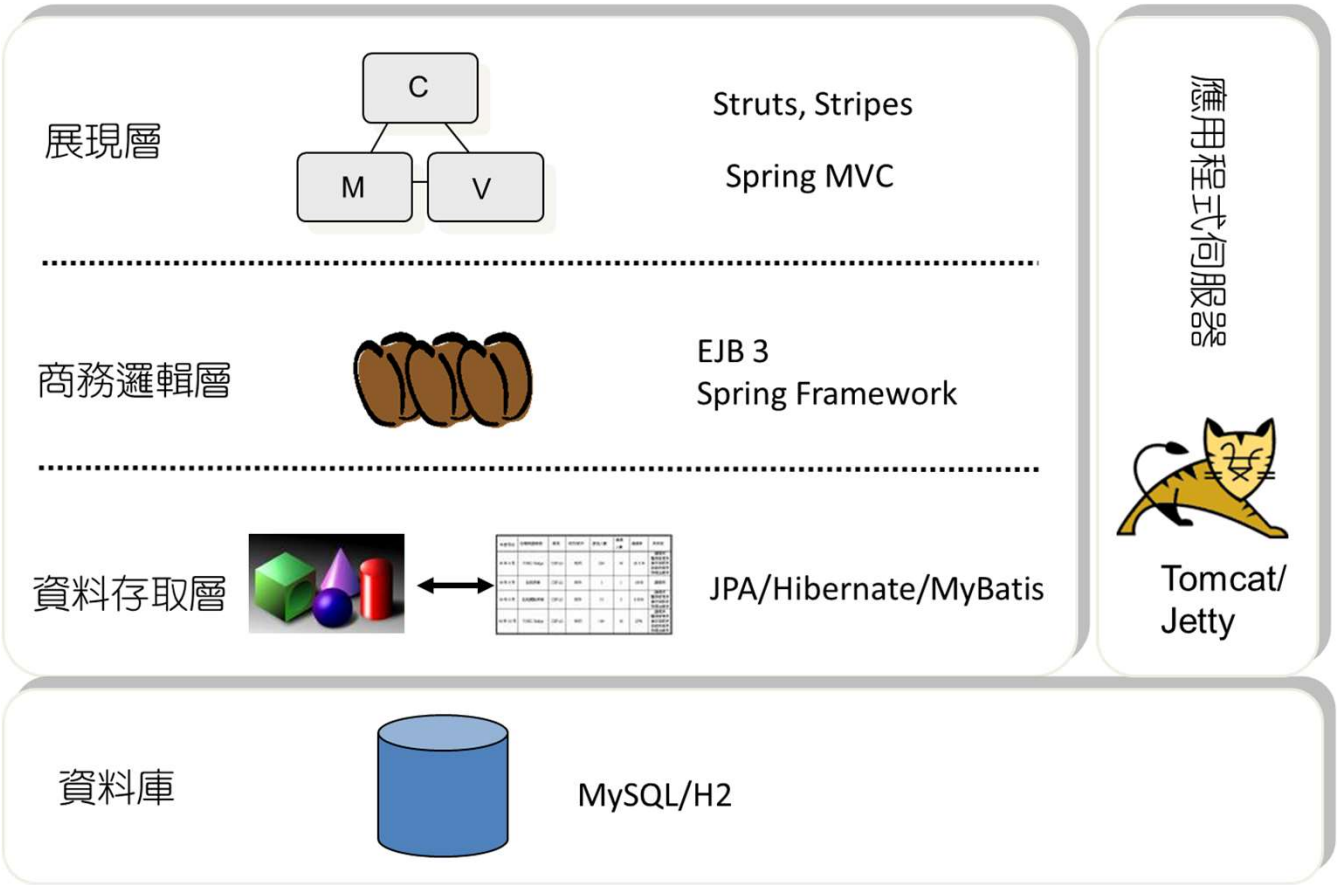
Step 4 寫作Client 程式

```
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



Advantages of Java EE

- Unique advantages
  - 長期實戰經驗的累積
    - Thousands of successful projects
    - 1998 ~ now
  - Reusing skills
  - Performance and reliability
    - More reliable, secure, and scalable
    - Complex and high-transaction are very well handled

Limitations of Java EE

- Complex application development environment
  - Can be difficult to understand for beginners
  - Many APIs to learn
- Development efforts
  - The final cost of a project including development and deployment
  - Application development may end up being expensive
- Testing
  - Hard to test without containers
  - Some solutions exist: https://arquillian.org/

Spring Framework

- 核心: 基於Dependency Injection的框架
  - 基於DI核心，發展出一系列相容於Java EE的框架
    - Spring MVC: 基於Servlet
    - Spring Data: 基於JDBC/JPA
    - Spring Boot: 內嵌Web container
      - Tomcat or Jetty
  - 不相容的部份
    - EJB
    - JAX-WS/JAX-RS
- 影響
  - Java EE 5之後，Java EE受到Spring與Hibernate很大影響
    - RI與CDI慢慢取代JNDI
    - EJB 3 POJI/POJO based programming model
    - 以JPA取代CMP

Spring Framework的設計理念

- 可模組化的輕量級服務Lightweight DI container
  - 一致的設定檔格式
- 透過介面進行系統設計
  - Programming by interface
- 以傳統Java物件為主
  - POJO-based Programming

Spring Boot

- 自動產生各式企業應用程式樣板的懶人包
  - 特別適合Cloud Native環境與Microservice架構
    - 自我包含，容易於Container運行
    - 不依賴AP Server
    - 依賴maven或gradle
  - 重要理念
    - 自我包含，內嵌server
      - 類似: Fat jar, Express.js
    - Convention-over-configuration
      - 類似: Scala
    - 內建observability
      - Spring actuator
    - 套件管理
      - 類似: npm

Benefits of using Spring framework

- Simple and easy to test
  - POJO and AP server free
- DI
  - Making components loosely coupled
  - It can be used to efficiently organize middle-tier objects
- Well-developed MVC framework
  - JEE沒有提供特定MVC Framework
  - Spring MVC

Disadvantages of Spring

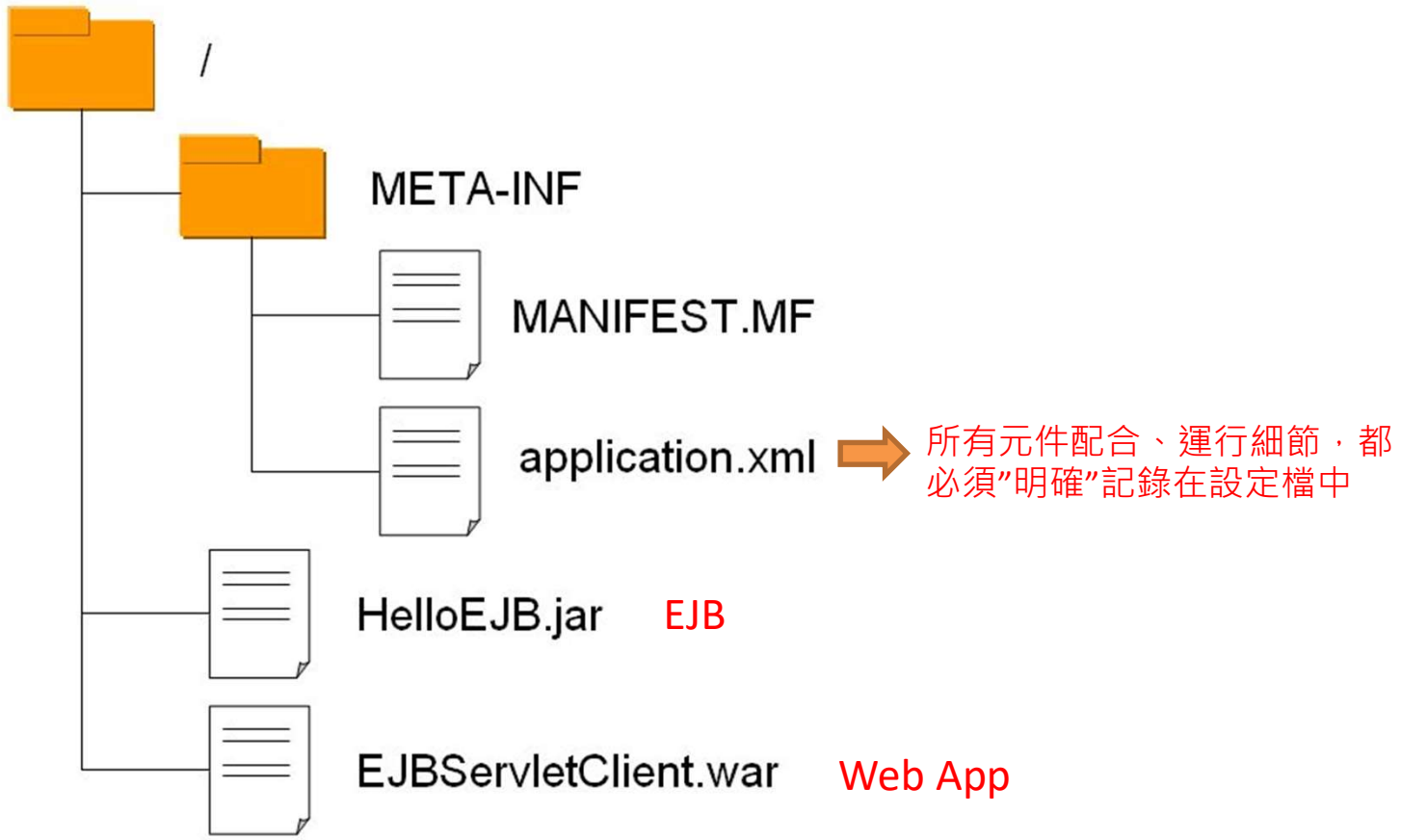
- DI
  - 屬於比較進階的技巧
  - 對OO不熟悉的人可能會有點困惑
- Wiring hell
  - Must have some knowledge of XML
  - Clear guidelines on several topics are not present in Spring documentation
  - It takes a lot of time and effort in initial configurations
  - Dependency management can be a huge burden

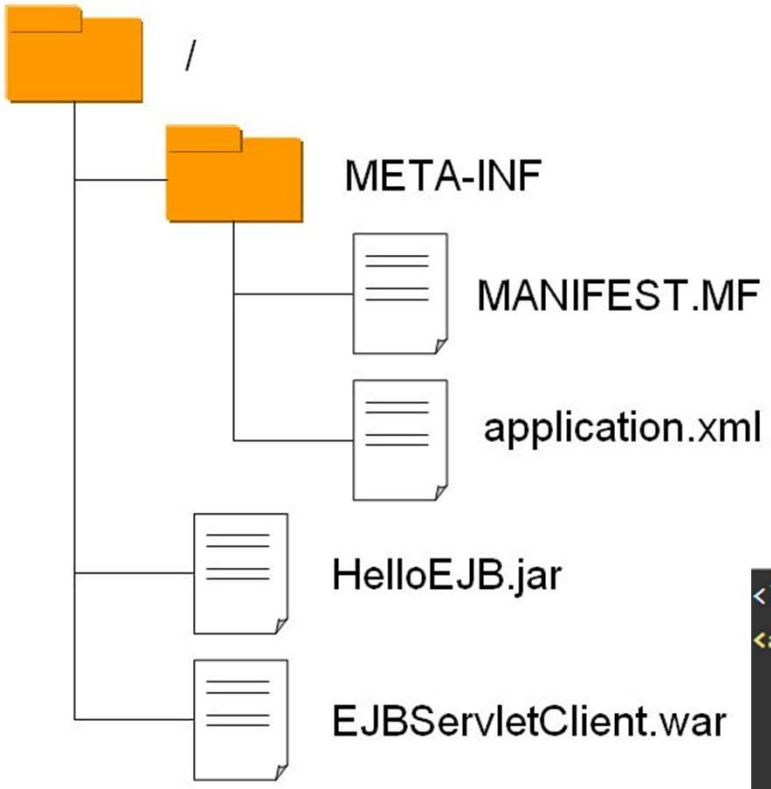
建議

- Spring與Jakarta EE技術上並不互斥
  - 可以混用
  - 二者互相影響
    - 就歷史上來看，Spring是沿著Java EE規格在發展
    - JEE的CDI事實上也是Spring DI精神的實現
    - Java EE在2006之後，也受Spring與其它OSS Project影響很大: EJB3、JPA、CDI、RI
    - Spring boot: convention over configuration and sensible default philosophies最早是Java EE Ap server的哲學
  - Spring母公司vmware 也是 Jakarta EE community成員
    - https://www.eclipse.org/membership/exploreMembership.php

傳統AP Server做法

- EAR

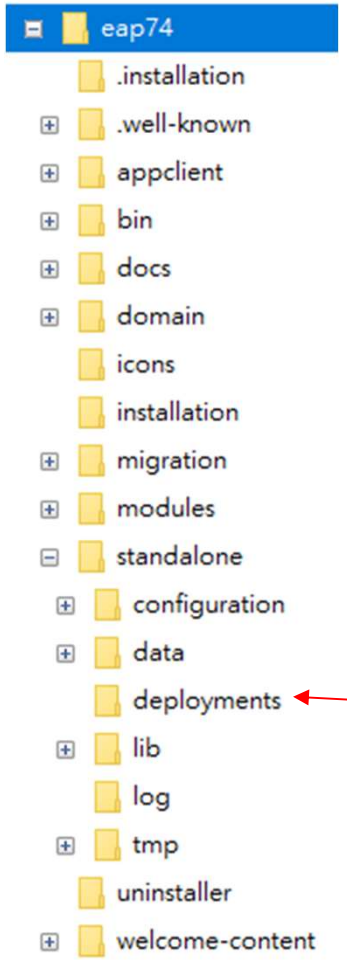


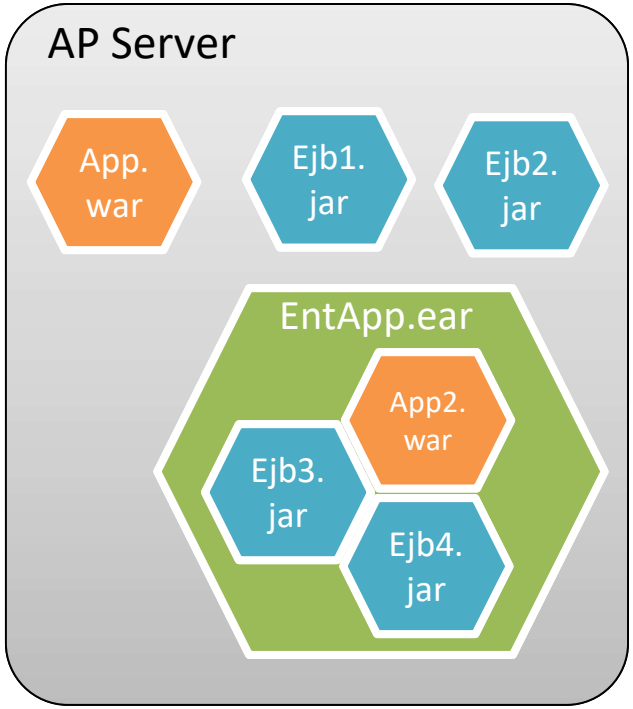


從設定檔就可以大致了解程式如何運行

```xml
<?xml version="1.0" encoding="UTF-8"?>
<application version="5" xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                   http://java.sun.com/xml/ns/javaee/application_5.xsd">
  <display-name>HelloApplication</display-name>
  <module>
    <web>
      <web-uri>EJBServletClient.war</web-uri>
      <context-root>/EJBServletClient</context-root>
    </web>
  </module>
  <module>
    <ejb>HelloEJB.jar</ejb>
  </module>
</application>
```

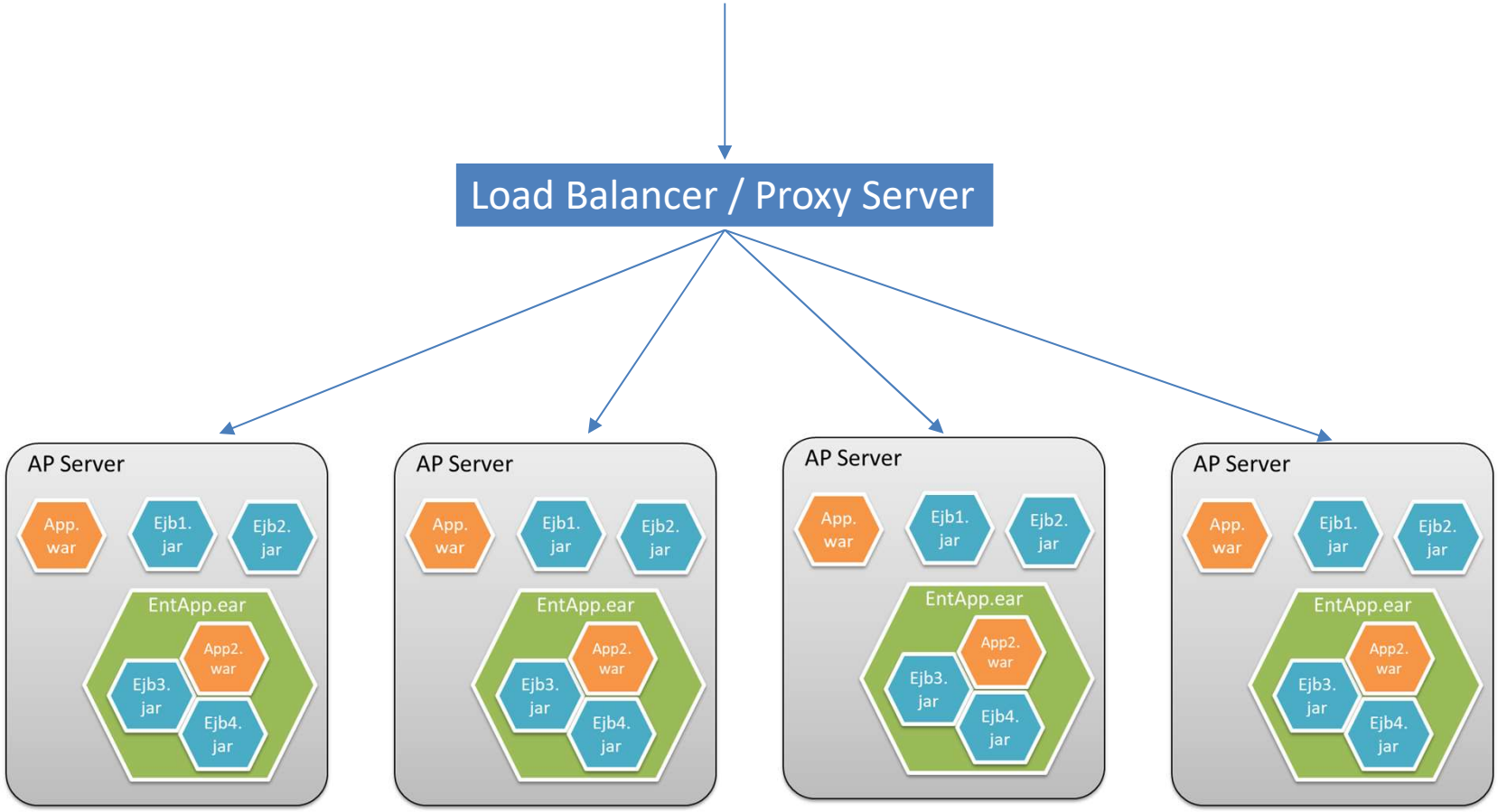
Ex: Jboss EAP 7.4





所有ear, war, jar都放在這裡

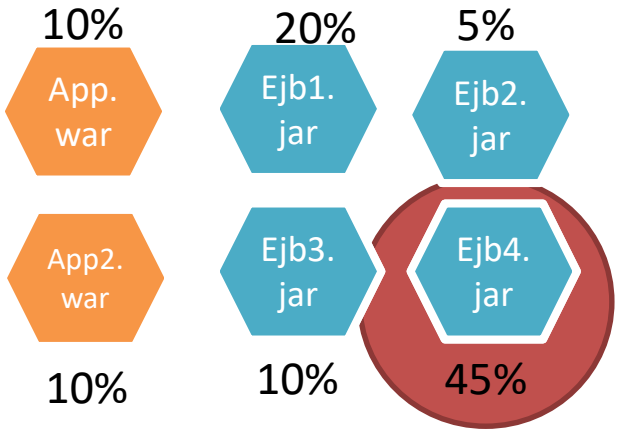
Horizontal Scaling of AP Server



這樣做有什麼問題嗎?

Horizontal Scaling of AP Server

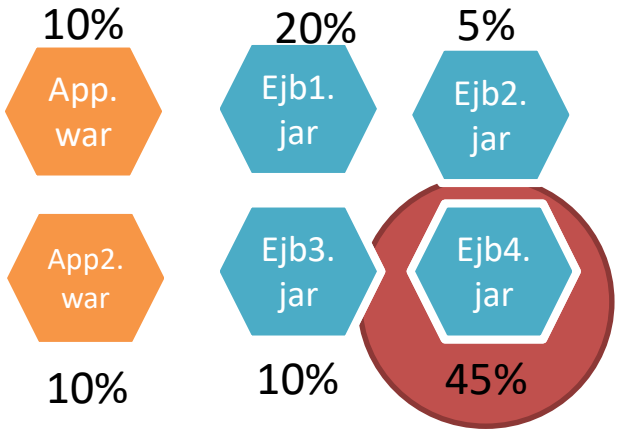
假設這些元件有下列負擔比例

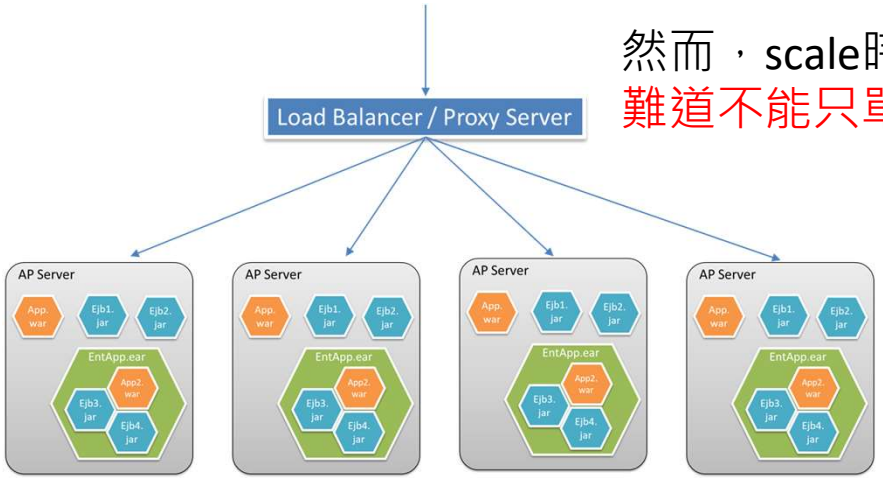


很明顯，EJB4.jar需求度遠大於其它元件

Horizontal Scaling of AP Server

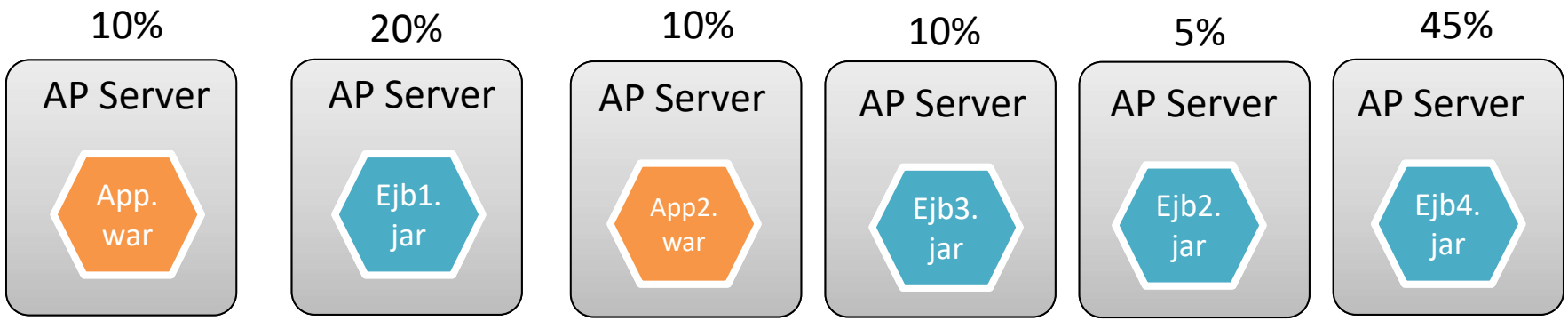
假設這些元件有下列負擔比例



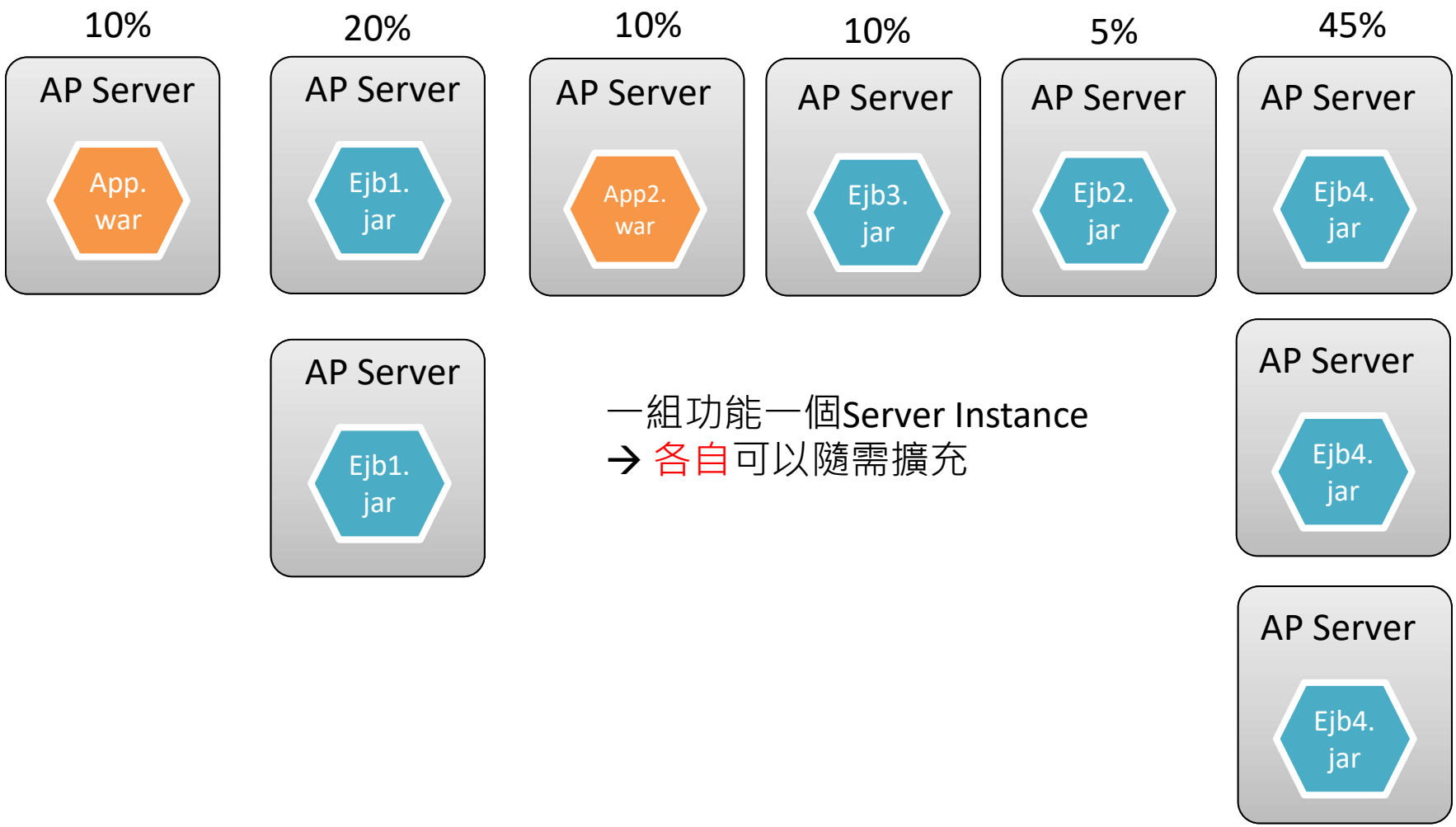


然而，scale時，是大家要一起複製

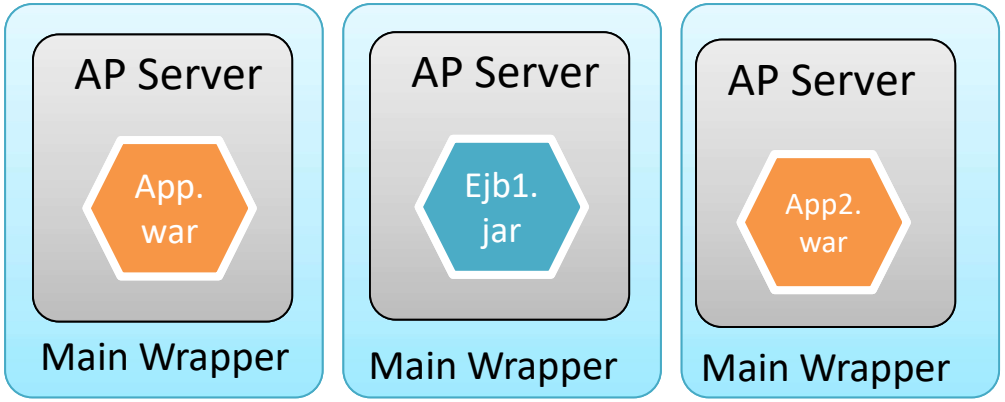
難道不能只單獨scale EJB4.jar嗎?



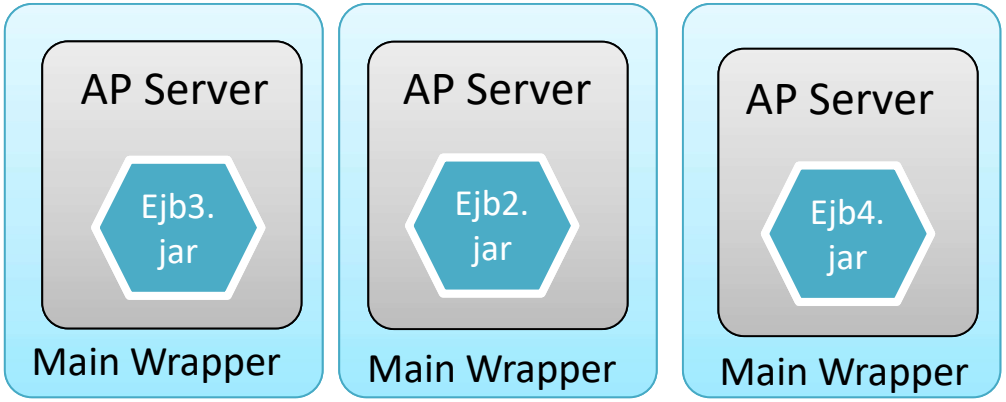
一組功能一個Server Instance
→ 各自可以隨需擴充


10%
AP Server
App.
war
20%
AP Server
Ejb1.
jar
AP Server
Ejb1.
jar
10%
AP Server
App2.
war
10%
AP Server
Ejb3.
jar
5%
AP Server
Ejb2.
jar
45%
AP Server
Ejb4.
jar
AP Server
Ejb4.
jar
AP Server
Ejb4.
jar
一組功能一個Server Instance
→ 各自可以隨需擴充

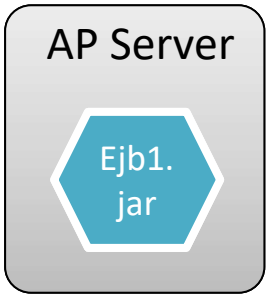
Embedded Server



透過Main Wrapper
整包可做為可執行檔直接執行
例如
Java –jar app2-war.jar



Embedded Server



- 定義
  - AP Server是應用程式的一部份
    - Application和AP Server打包為一個單位
    - AP Server以類別方式存在，透過物件操作來設定與啟動
  - 為什麼要Embedded Server
    - 易於部署與啟動
      - 主部打包成一個jar; 用java –jar xxxx.jar就可啟動
      - 特別適合container（ex: Docker）
      - 配合K8S/Docker，以細粒度隨需伸縮
    - 容易測試
      - 寫Test時，AP Server視為Test fixture直接操作

The Executable Jar File Structure

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-BOOT-INF
    +-classes
    |  +-mycompany
    |     +-project
    |        +-YourClasses.class
    +-lib
       +-dependency1.jar
       +-dependency2.jar
```

- Application classes should be placed in a nested BOOT-INF/classes directory
- Dependencies (用到的函式庫) should be placed in a nested BOOT-INF/lib directory

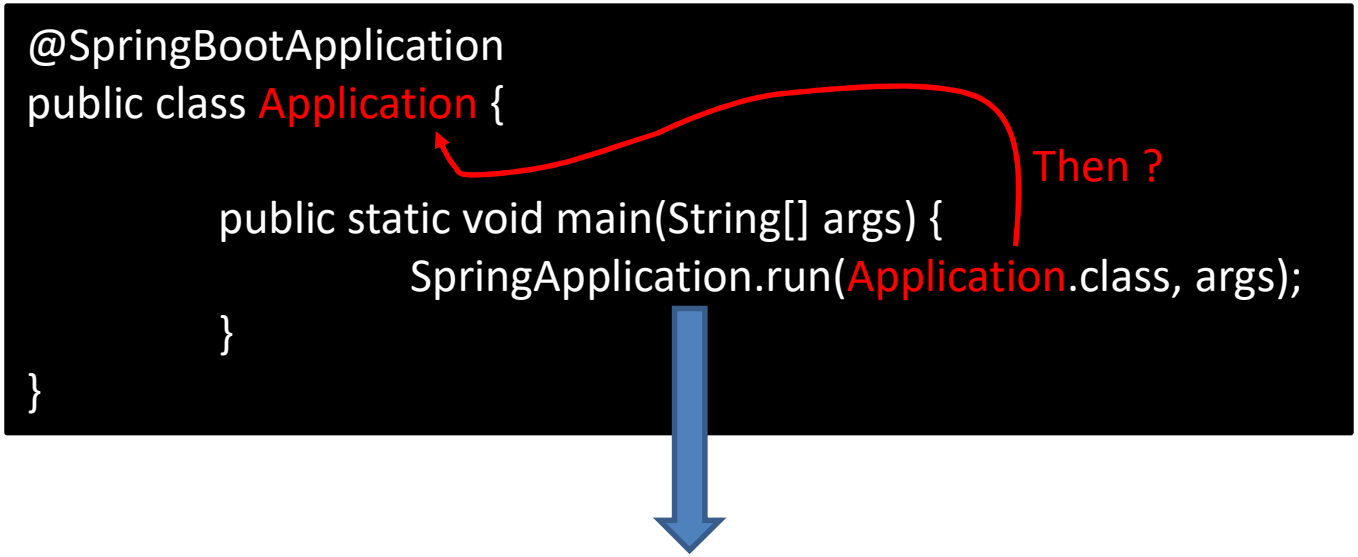
https://docs.spring.io/spring-boot/docs/2.0.0.M6/reference/html/executable-jar.html

The Executable War File Structure

```
example.war
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-WEB-INF
    +-classes
    |  +-com
    |     +-mycompany
    |        +-project
    |           +-YourClasses.class
    +-lib
    |  +-dependency1.jar
    |  +-dependency2.jar
    +-lib-provided
       +-servlet-api.jar
       +-dependency3.jar
```

- Application classes should be placed in a nested WEB-INF/classes directory
- Dependencies (用到的函式庫) should be placed in a nested WEB-INF/lib directory

required when running embedded but are not required when deploying to a traditional web container



1. SpringApplication.run(Application.class, …)
2. Application is this class
3. Then ?

重點在這裡:

@SpringBootApplication =

@Configuration +

@EnableAutoConfiguration +

@ComponentScan

我怎麼知道這個?

```
@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }
}
```

- @Configuration
  - 這個類別是Configuration Class，可以用來取代bean-config.xml
- @EnableAutoConfiguration
  - 系統會很智慧地由classpath中包含的類別的推論這個應用程式的類型，掛上相關beans並自動初始化。
  - 例如: 如果系統中有spring-mvc相關類別，它會自動掛上DispatcherServlet
- @ComponentScan
  - 系統會自動掃描具有Spring annotation的classes，並放到bean context中

```
@RestController
public class HelloController {

        @GetMapping("/")
        public String index() {
                return "Greetings from Spring Boot!";
        }

}
```

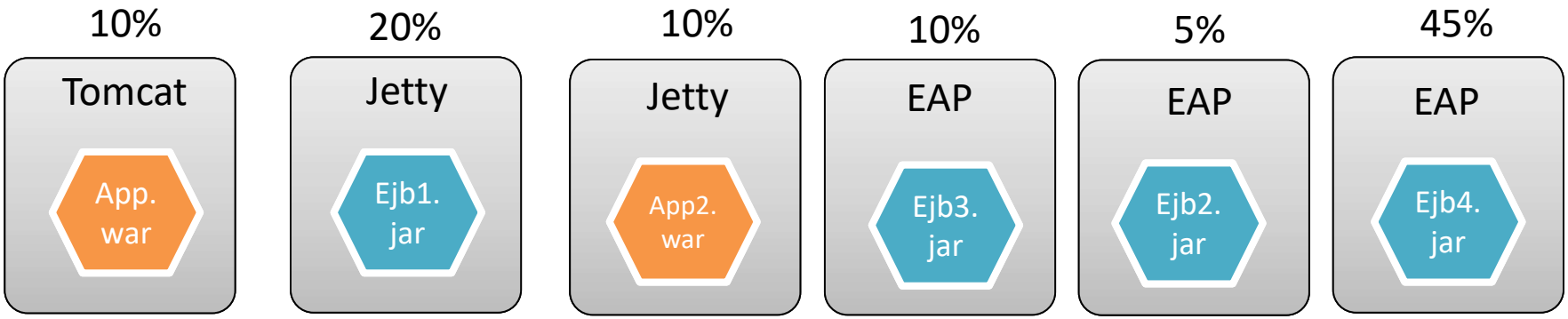
問: 無法從程式碼、設定檔中明確得知程式設定與組合方式?

答: 這就是Convention over Configuration 的精神

假設開發人員知道”Convention慣例” ←隱性知識

Embedded or Not?

- Pros
  - 細顆粒的隨需伸縮
  - Isolation
    - 每個微服務AP Server參數可以各自調整，甚至於可以是不同品牌的AP Server
    - 配合Docker (LXC)，甚至於可以是不同OS、不同jvm版本



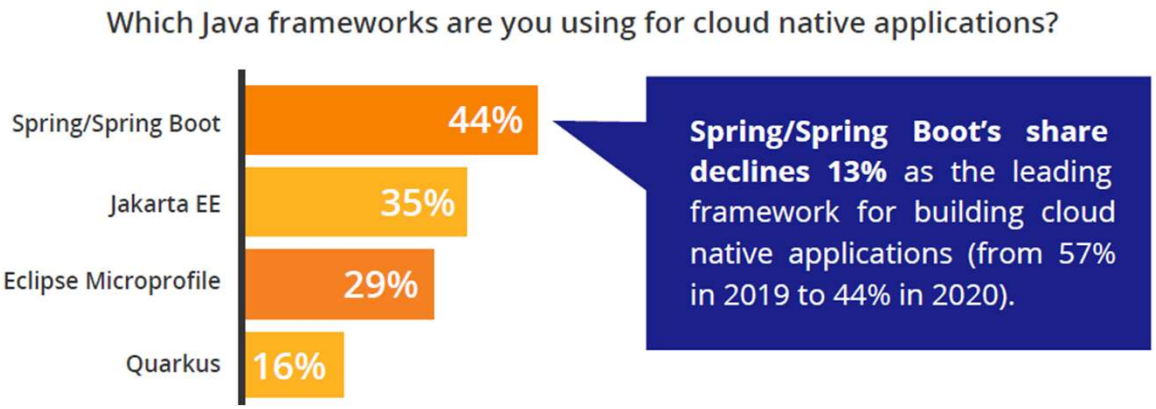
Embedded or Not?

- Cons
  - 維運負擔
    - Multiple versions to manage and monitor
  - 修改參數要重新建構原始碼
    - It's not possible to change properties files only without rebuilding
  - No hot deployment out of the box

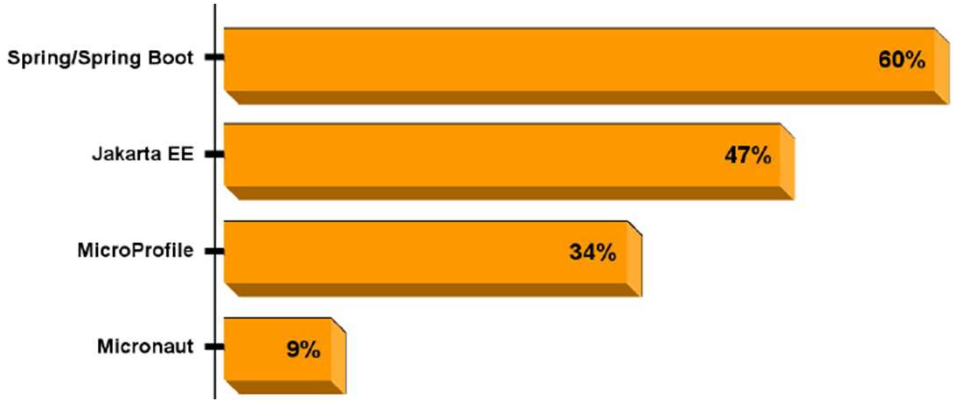
Spring Boot or Java EE ?

- 使用Spring Boot之後的好處
  - 大幅降低設定用程式碼
  - 只需要寫作/維護必要的邏輯
- 代價
  - Java EE中，所有設定均明確列出，可追朔
  - Spring Boot光靠程式碼與設定檔無法了解系統行為
    - Convention over configuration
    - 系統運行之後，在沒有文件及人員訓練不足的情況下，程式無法維護

統計數據

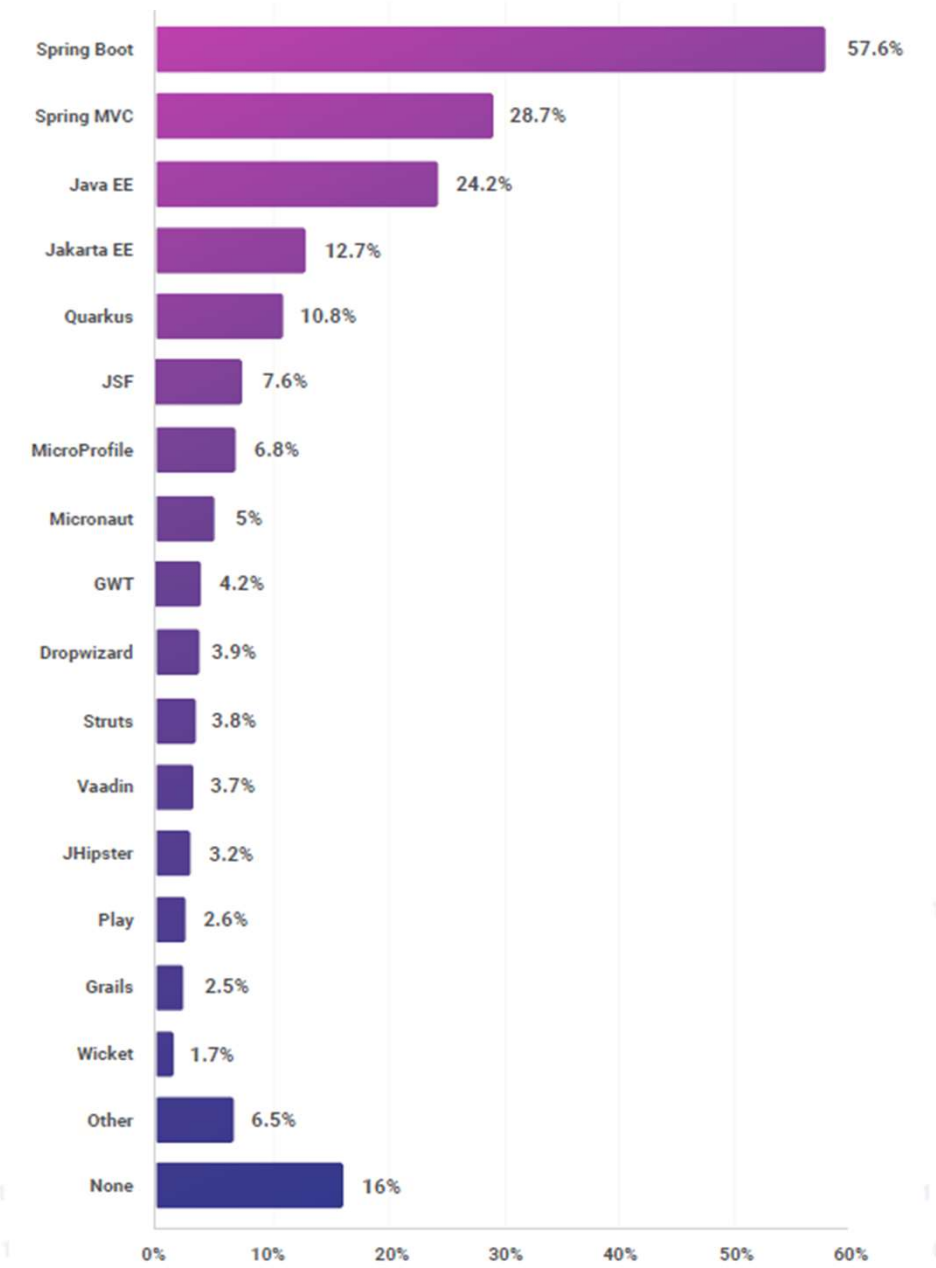


2020



2021

2020 and 2021 Jakarta EE Developer Survey Report

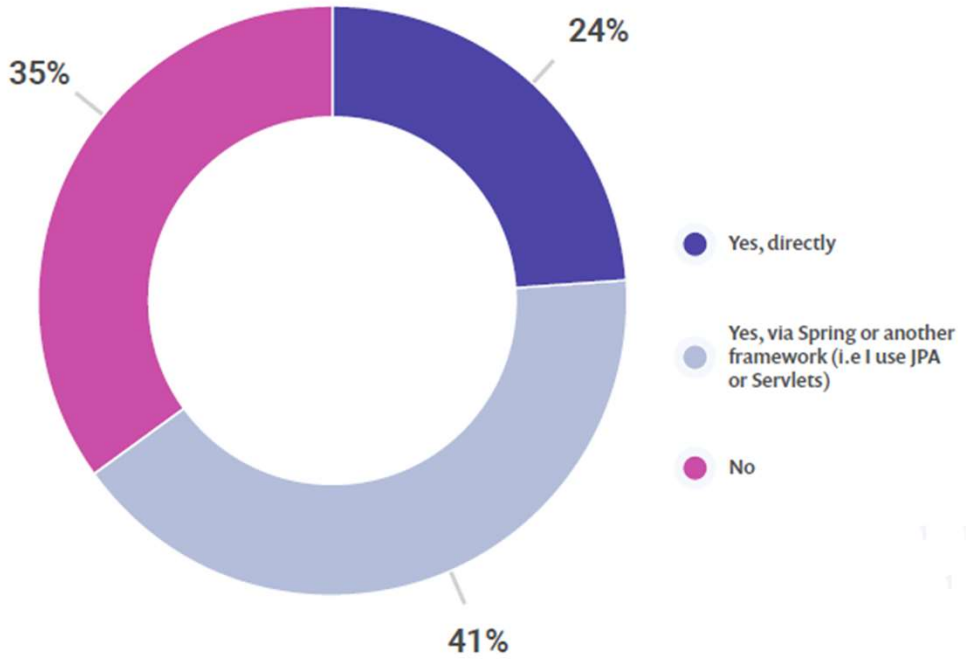

Spring Boot 57.6%
Spring MVC 28.7%
Java EE 24.2%
Jakarta EE 12.7%
Quarkus 10.8%
JSF 7.6%
MicroProfile 6.8%
Micronaut 5%
GWT 4.2%
Dropwizard 3.9%
Struts 3.8%
Vaadin 3.7%
JHipster 3.2%
Play 2.6%
Grails 2.5%
Wicket 1.7%
Other 6.5%
None 16%
0% 10% 20% 30% 40% 50% 60%

JVM2021

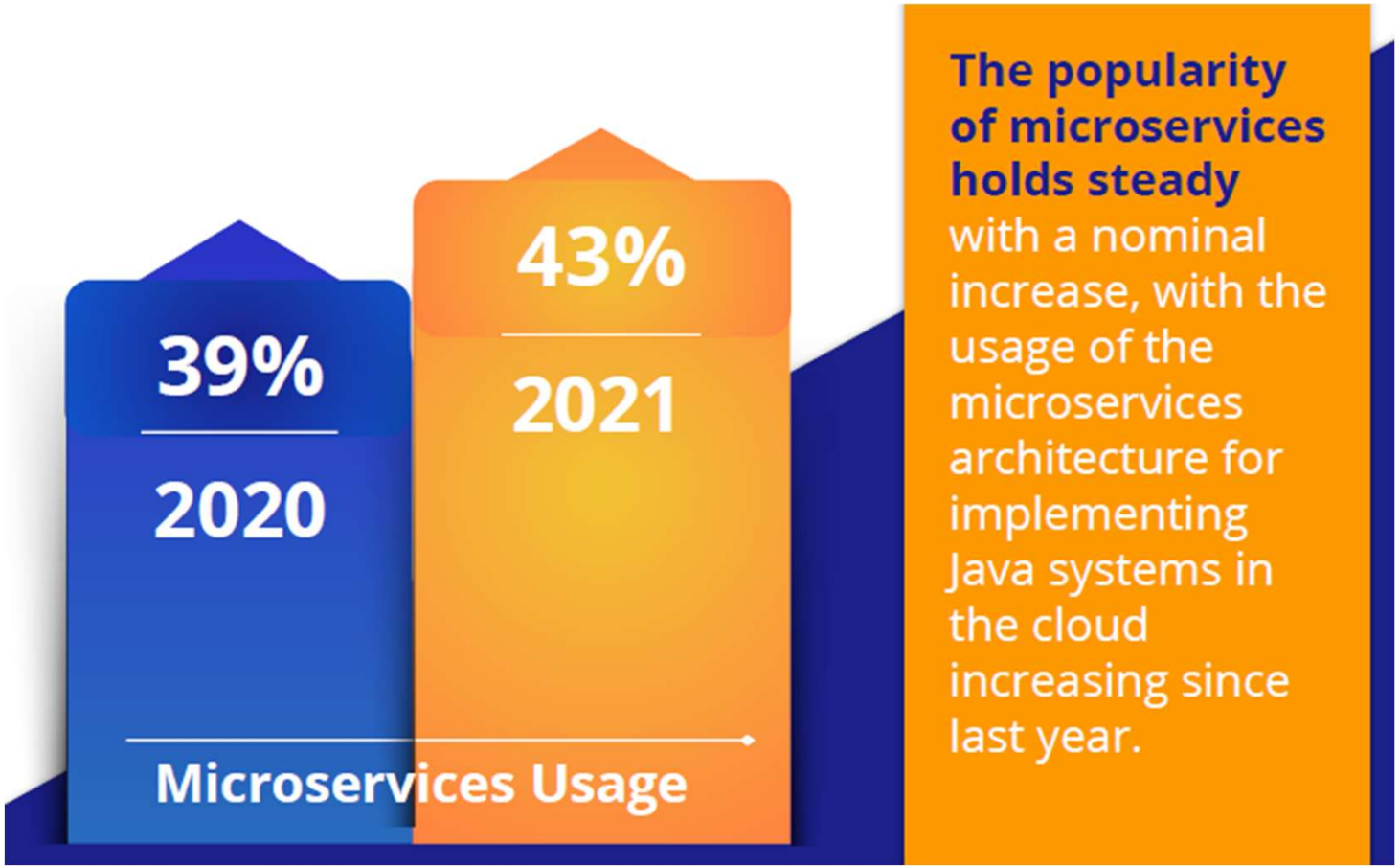
18. Do you use Enterprise Java? (J2EE, Java EE, Jakarta EE)

The question of whether Java developers use the Enterprise Edition (EE) of Java is something we ask every year. Only this year, we slightly changed the question. We added the option "Yes, via Spring or another framework (e.g. JPA or Servlets) to ensure that people who use EE indirectly do not choose the "No" option.

With 35% of developers reporting that they don't use Java EE, the landscape hasn't changed much since last year (38%). It is important to point out, however, that 4 out of 10 developers are using Enterprise Java indirectly. This does raise some concerns over Java EE's popularity.



微服務架構成長


39%
2020
43%
2021
Microservices Usage
The popularity of microservices holds steady with a nominal increase, with the usage of the microservices architecture for implementing Java systems in the cloud increasing since last year.

結論與建議

- 長期來看，Spring 可能會dominate整個Java企業端
  - 尤其在Cloud Native/微服務的環境
  - 傳統容器元件架構漸被系統容器平台取代(docker+k8s)
    - 多元件共享一個Server→一個元件一個Server並放在一個docker中
      - Spring Boot 的內嵌架構佔極大優勢; 未來Jakarta EE也以類似架構為主
- 就市佔率來說，Jakarta EE目前仍是第二大
  - 仍持續發展進步
  - 說Jakarta EE phase out言之過早
  - 迷思: Jakarta EE/Spring只能二選一

Q & A