元件圖與部署圖

- 在UML中，元件圖(Component Diagram)以及部署圖(Deployment Diagram)是用來呈現系統實作的模型。所謂的實作包含了各軟體元件的結構以及執行時(run-time)的元件佈署情形。元件圖所呈現的是元件之間的關係與的結構。佈署圖則呈現出在實體上各節點間的結構，以及各元件在節點中的配置佈署。

元件圖

- 在UML 2.0中，元件圖（Component Diagram）的定義是：以模組化的方式塑模系統的靜態結構，此模組化結構封裝了系統的內容，並且該模組化結構是可以在其所處的環境中被替換。

元件圖

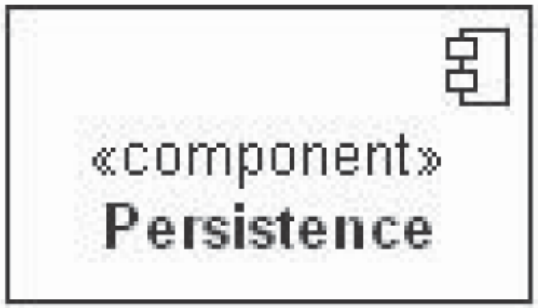
- 元件圖的主要目的是以模組化的方式來呈現系統中，軟體組成部分的組織結構以及它們之間的相依關係；並且，為了讓元件可以達到被替換的目的，元件使用了「必須的」（Required）以及「提供的」（Provided）兩種介面來定義其行為。

元件圖

- 元件
  - 在UML 1.x中，元件（Component）圖是以一個長方形加上兩個向外凸 出的矩形來表示。從UML 2.0開始，元件圖只使用一個長方形來表達，圖中可以使用造型<<component>>用以標示出此長方形為一個元件；另一方面， 也可以將之前在UML 1.x中的元件圖則移至長方形的右上角，顯示成一個小 圖案，但這個小圖案的繪製並不是必須的，圖15.1顯示一個元件圖。

元件圖

- 元件
  - 每個元件圖擁有唯一的元件名稱，元件名稱基本上只是字串，用以表示 此元件在系統中所代表的組成部分，例如說在系統中，我們要表達一個負責 管理資料庫存取動作的元件Persistence，那麼我們可以利用元件圖表示如圖15.2。

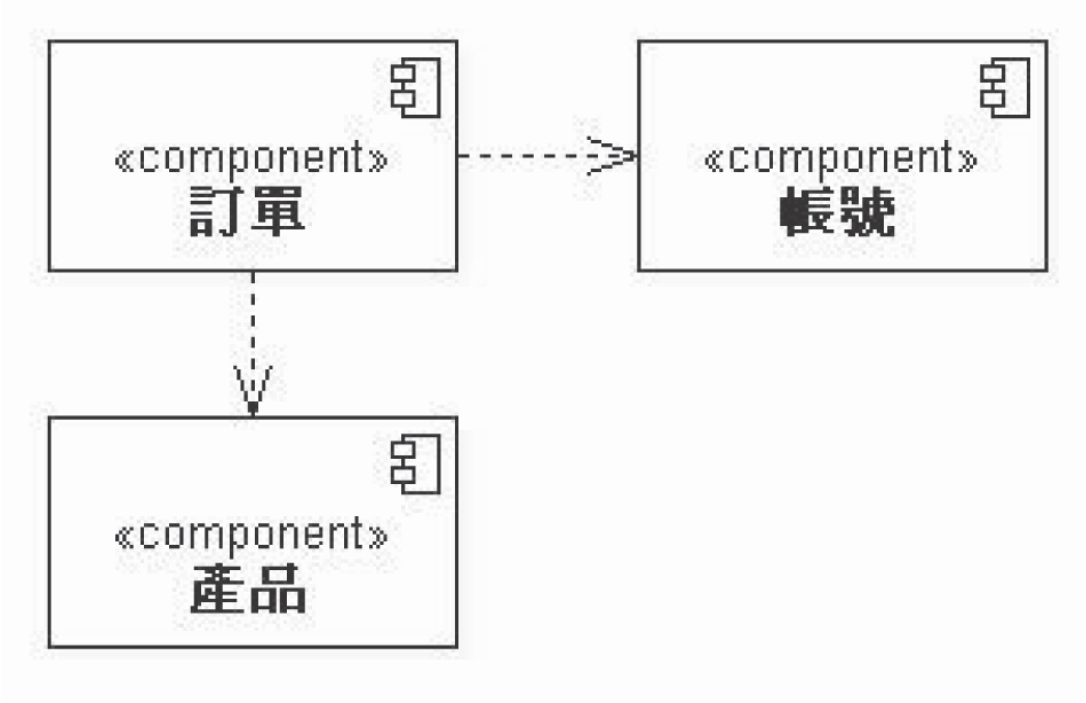


元件圖

- 元件
  - 一個系統是由許多不同的元件所組成，而元件之間可能具有某種程度的相依性。對於元件之間的相依性可以採用與類別圖之相依性關係（Dependence）來表達。例如說一個購物系統是由訂單（Order）、產品（Product）、以及帳號（Account）等三個元件所組成則我們可以繪製如圖15.3之元件圖來表示它們之間的相依性。

元件圖

- 元件



元件、介面、類別

- 對於元件的塑模可以使用兩種不同的方式來著手：第一種方式是從外部的觀點（External View），第二種方式是從內部的觀點（Internal View）。從外部的觀點來看元件是把元件看成是一個黑箱（Black-box），主要的塑模 重點放在元件所「提供的」或是「需要的」介面上；從內部的觀點來看元件是把元件看成是一個白箱（White-box），主要的塑模重點是元件內部的類別以及它們與介面的關連性。

元件、介面、類別

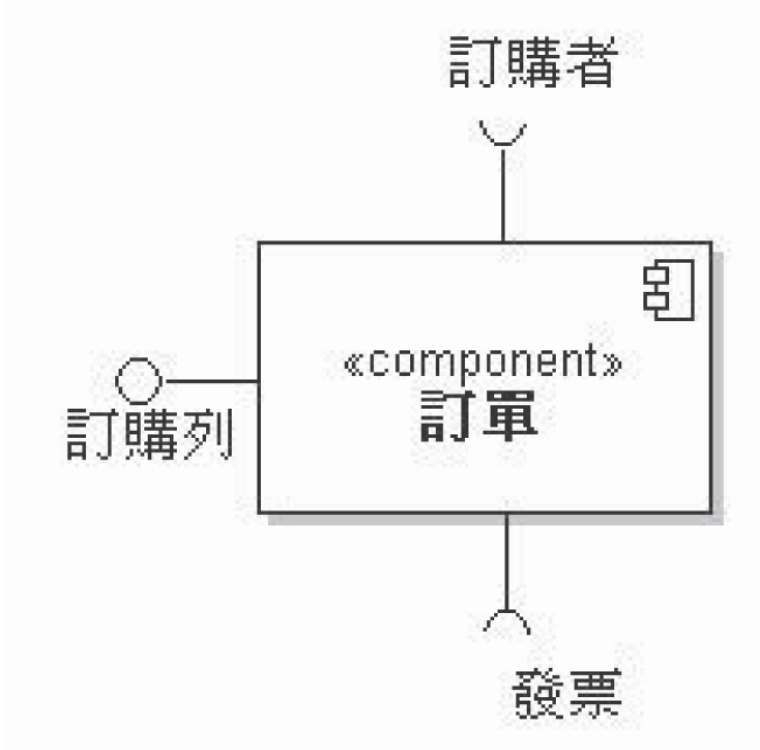
- 元件外視圖
  - 元件與介面之間的關係相當密切，以元件為基礎的（Component-based）設計像是Microsoft的COM、DCOM或是Sun的EJB（Enterprise Java Bean）等，都是利用介面的機制來將所有的元件連繫在一起。介面定義操作而不實作它，而元件中則包含了實作這些介面的具體類別。一般來說， 一個元件中會包含許多實作介面的類別。

元件、介面、類別

- 元件外視圖
  - 如前所述，元件外視圖（External View）把元件看成是一個黑箱，其主要目的在於塑模介面。元件圖中表達的介面分為兩種：需要的（Required）以及提供的（Provided）。顧名思義，所謂「需要的」介面代表著該元件 所依賴的介面；而所謂「提供的」介面代表該元件所提供的功能。需要的介面以一個半圓形與元件相連，提供的介面則是以一個小圓圈與元件相連。圖15.4顯示一個「訂單」元件，該元件提供一個「訂購列」介面，另外，圖中亦顯示出「訂單」元件需要使用到「訂購者」與「發票」兩個介面。

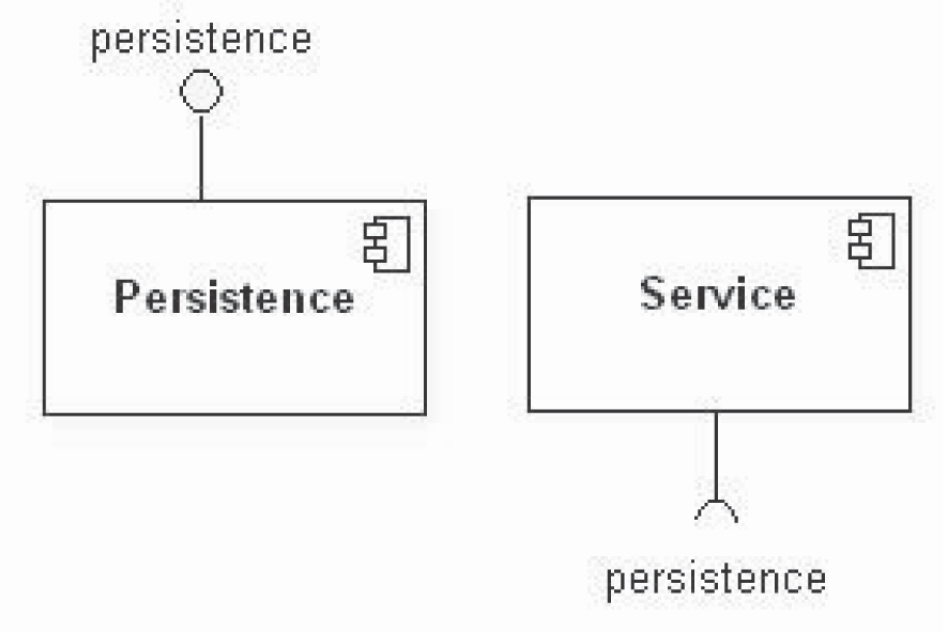
元件、介面、類別

- 元件外視圖



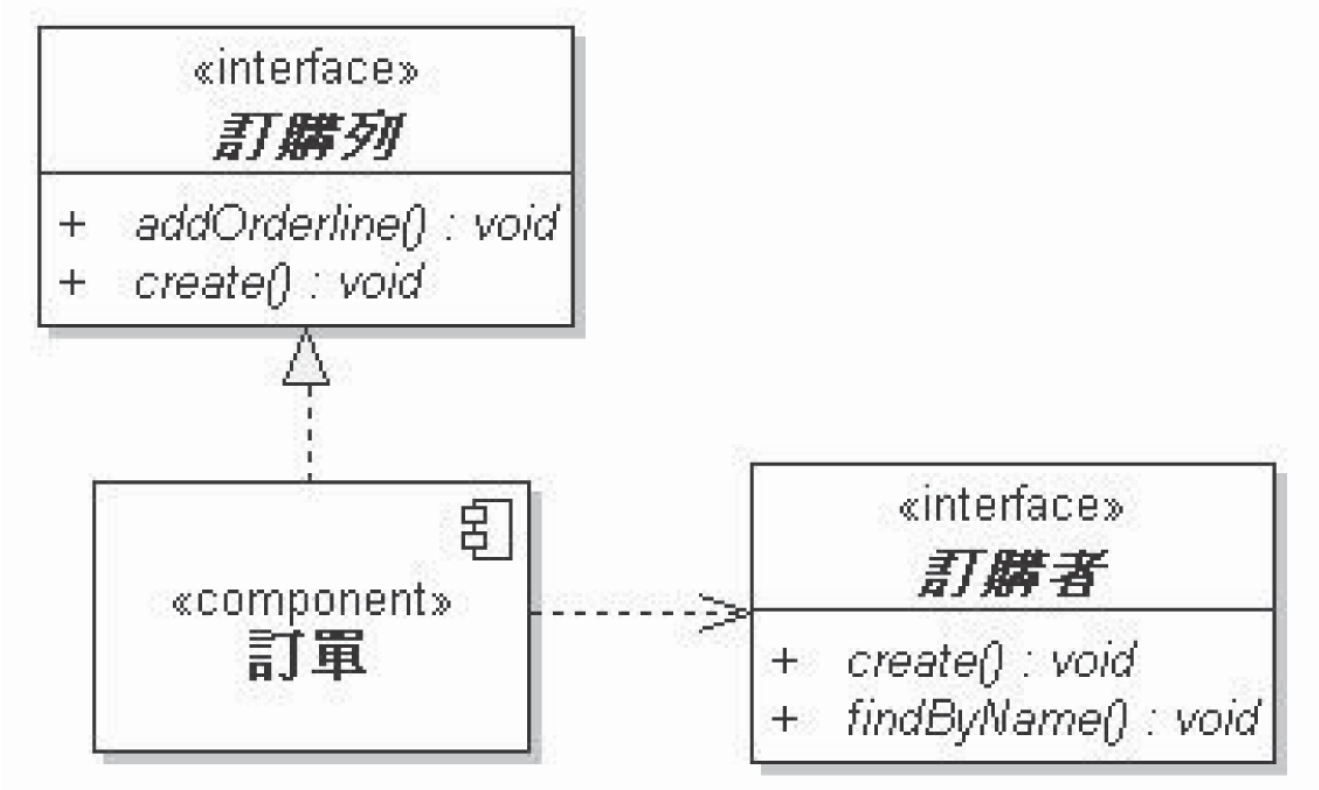
元件、介面、類別

- 元件外視圖
  - 圖15.5則塑模兩個元件：Persistence以及Service；其中，Persistence元 件提供了persistence介面，而Service元件需要使用到persistence介面。



元件、介面、類別

- 元件外視圖

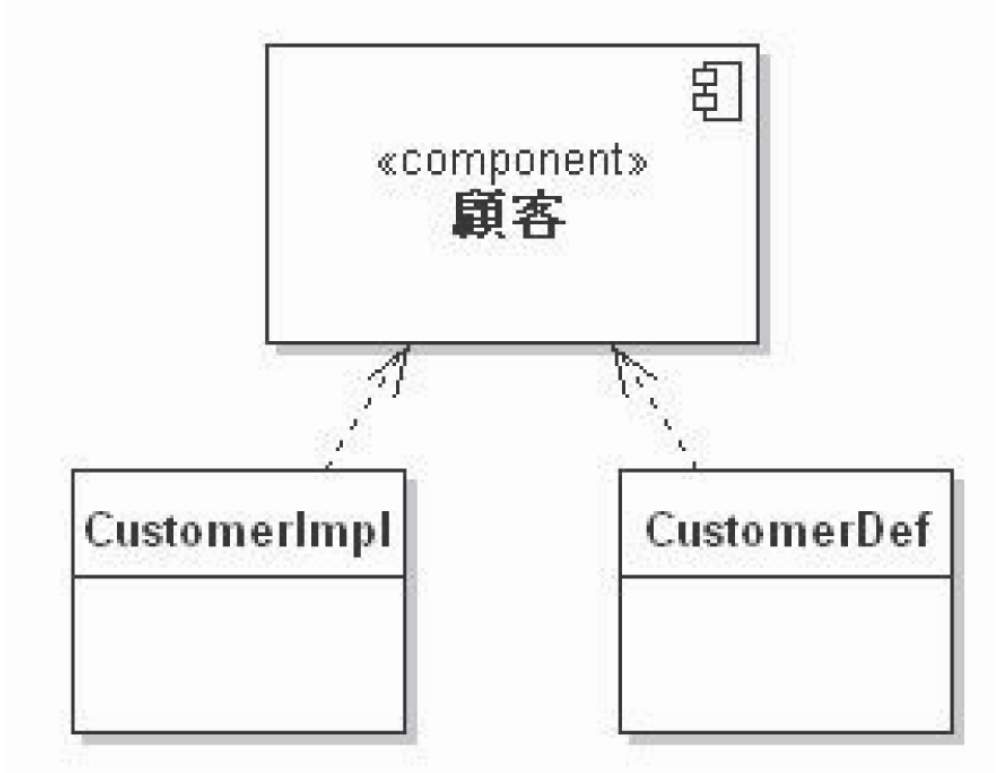


元件、介面、類別

- 元件內視圖
  - 元件內視圖（Internal View）是從內部的觀點來看元件，把元件看成是一個白箱，其主要的目的在塑模元件內部的類別以及它們與介面的關連性。在元件的內部視圖中使用與相依關係相同的圖形來塑模實作元件介面的類別。例如圖15.7顯示出類別（CustomerImpl）以及分別具體化顧客（CustomerDef）這個元件的介面。注意到這個關係並不是相依（Dependence）而是實作（Manifestation）。

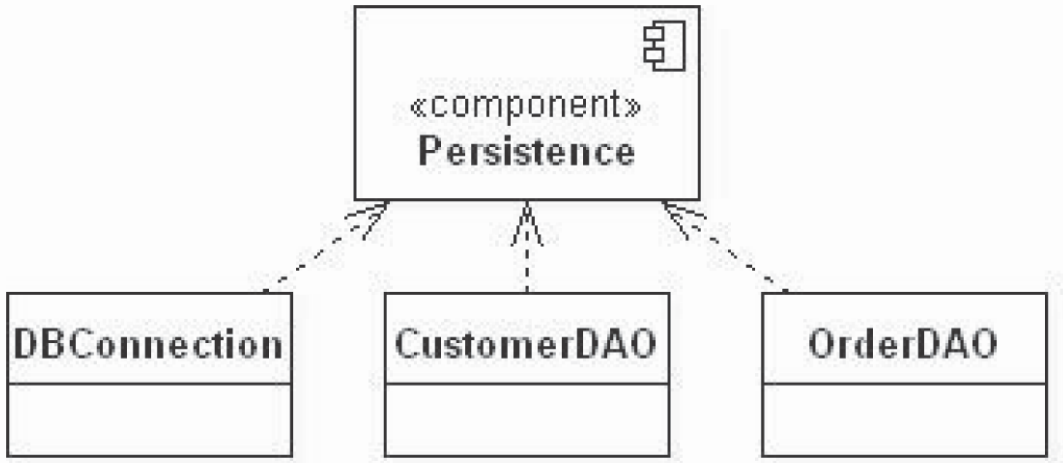
元件、介面、類別

- 元件內視圖



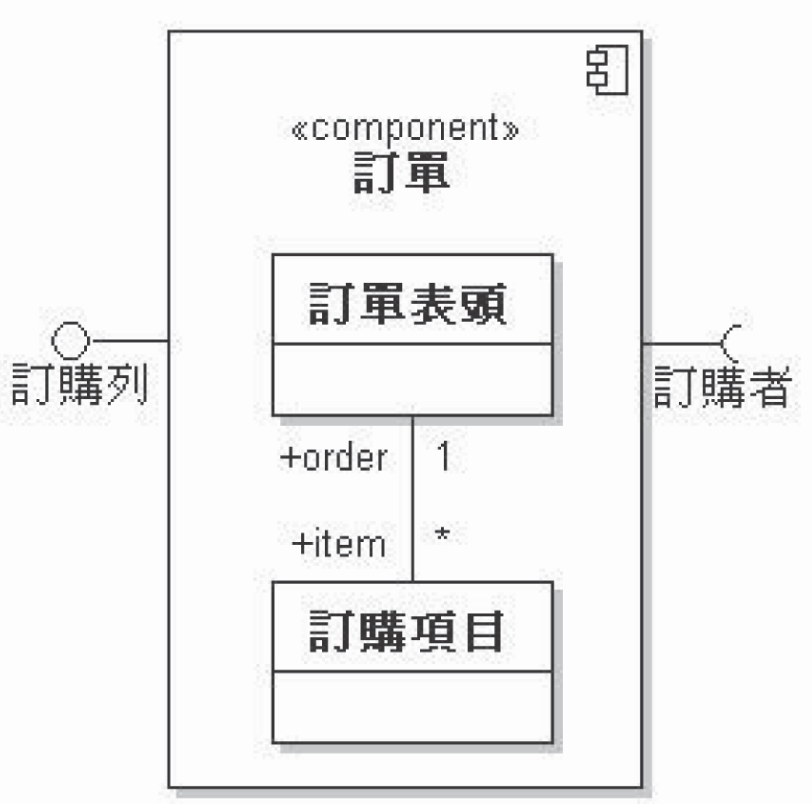
元件、介面、類別

- 元件內視圖
  - 範例說明：以之前所提到的Persistence元件，如果它所提供的介面是由DBConnection，OrderDAO，CustomerDAO這三個類別來實現的話，那麼我們可以利用元件圖來表達出它們之間的關係，如圖15.8所示。



元件、介面、類別

- 元件內視圖
  - 範例說明：在元件的內視圖中也可以使用巢狀的表達方式呈現元件的內部組成類別 以及類別之間的關係。例如圖15.9顯示「訂單」元件中包含了兩個類別「訂 單表頭」以及「訂購項目」， 並且此內視圖中也一併顯示出此兩個類別的一 對多關聯關係。

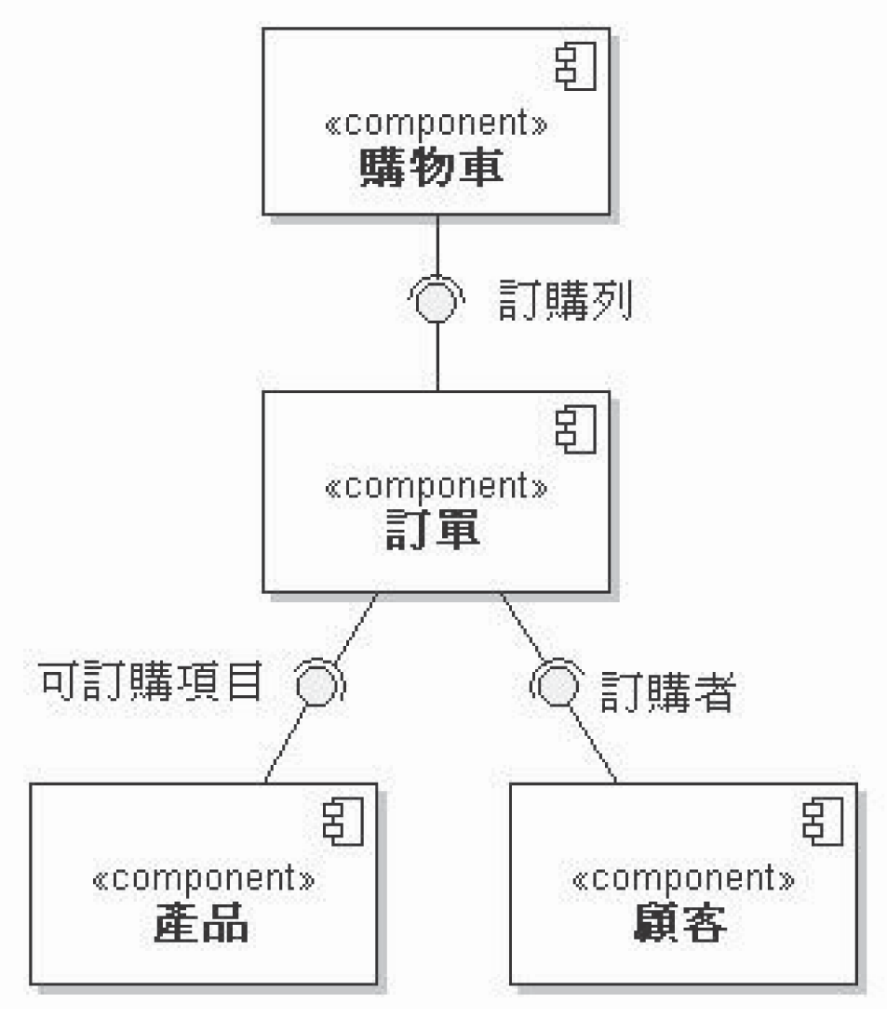


連結器

- 組裝連結器
  - 在元件圖中，有時候我們不只希望表達一個元件的外部或是內部結構而已，我們更希望能夠從元件圖中看出一個元件所提供的介面是被使用於哪一個元件。

連結器

- 組裝連結器

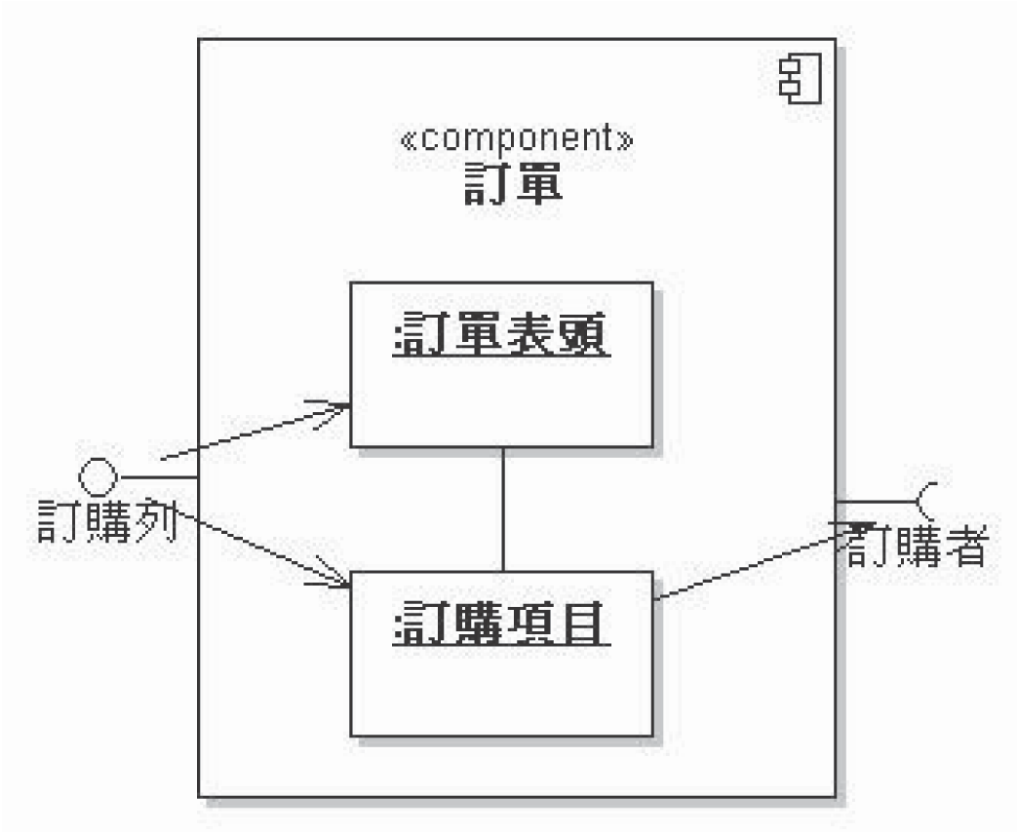

«component»
購物車
訂購列
«component»
訂單
可訂購項目
訂購者
«component»
產品
«component»
顧客

連結器

- 委派連結器
  - 我們知道在元件的內視圖中可以使用巢狀的表達方式來呈現元件的內部 實作類別。如果進一步地我們想要塑模哪一個介面與那個類別有關連的話， 那麼可以使用委派連結器來達成。委派連結器（Delegate Connector）的主 要目的是用來連結一個元件的外部介面至元件內部具體化該介面的物件；或 是連結元件內部物件至其所需的外部介面。委派連結器的圖形是一條帶有線 頭的直線，一端連接於介面端，另一端連結於元件的內部類別。例如圖15.11 顯示「訂購列」介面是由「訂單表頭」以及「訂購項目」來具體化；而訂購項目需要使用到「訂購者」介面。

連結器

- 委派連結器



部署圖

- 部署圖（Deployment Diagram）的主要目的是用來呈現系統開發工作 產出或是產出實體在節點的部署情形。所謂的工作產出指的是像文件、可執 行檔、相關檔案等等；而節點指的是硬體。換句話說，部署圖塑模系統的硬 體配置靜態觀點以及在各項硬體中所需部署的軟體元件。在部署圖中，節點 與節點之間以直線連結，用以表示它們之間有溝通的關聯性。

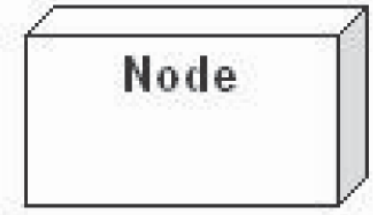
部署圖

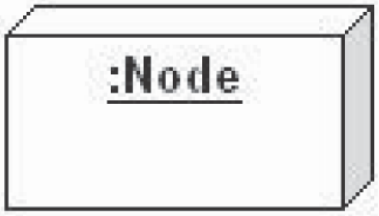
- 工作產出
  - Artifact代表在系統開發過程中任何工作上所製造出來的東西。系統開發 過程的產出很多，包括了需求分析文件、程式碼檔案（Source Code File）、函示庫（Library）、可執行檔（Executable file）、資料庫設計圖、資料庫、甚至是我們正在討論的UML圖等等。Artifact的表法如圖15.12所示，另外，在眾多的artifact型態中，為了表明該工作產出（Artifact）為何種物 件，可以在artifact中使用造型，例如<<file>>、<<document>>…等等。

部署圖

節點

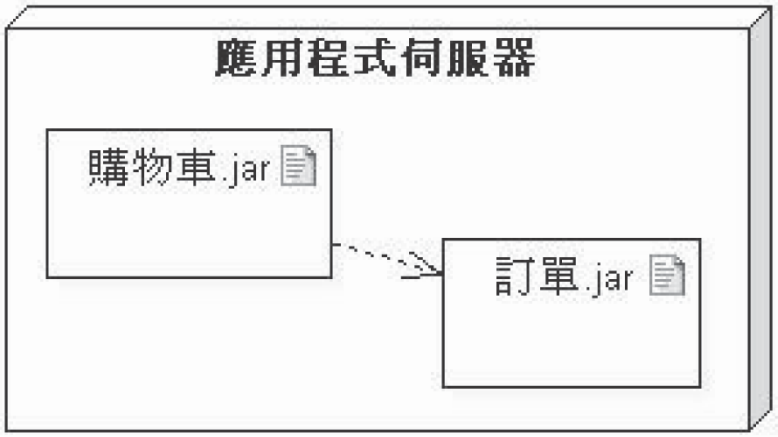
- 部署圖中的節點是指系統中有運算能力的硬體資源。節點（Node）的圖 形以一個立方體來表達，如圖15.13(a)；事實上，此圖表示的是一個節點的型態（type）。如果要表達的是某個特定的節點，則可以使用實體節點，如圖15.13(b)所示。這兩種節點的表法與「類別-物件」的概念相似，亦即是類別表達的是型態，物件則是類別的實體化。





部署圖

- 節點
  - 節點內部可以繪製部署於該節點之工作產出。例如圖15.14以一個節點 來表達一個應用程式伺服器，並且繪製出該節點所包含之購物車.ｊａｒ以及訂 單.jar兩個jar實體檔案。

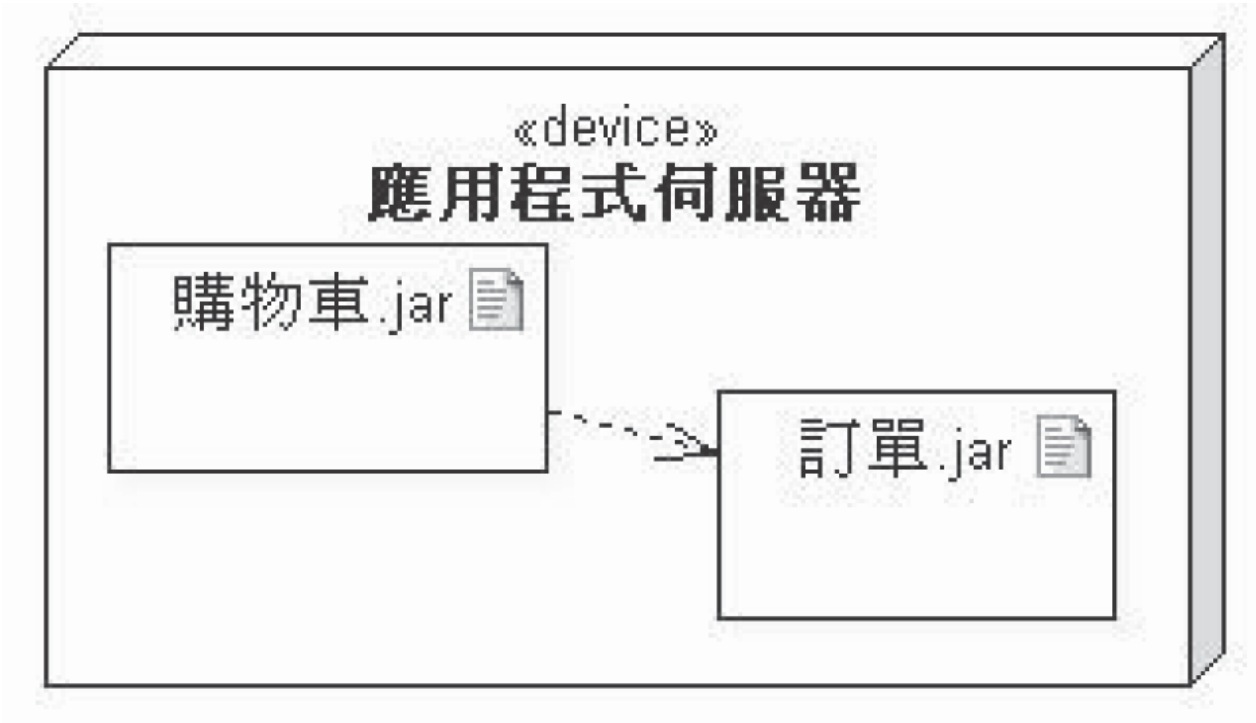


部署圖

- 節點
  - 對於節點，UML 2.0定義了兩個造型：<<device>>（裝置）以及<<execution environment>>（運行環境）。所謂的device是指一個具有處理能力的實際運算資源，並且artifact可以被部署於其上與執行。例如圖15.14中的應用程式伺服器節點就符合此定義，因此可以加上<<device>>造型於節點上，如圖15.15所示。

部署圖

- 節點

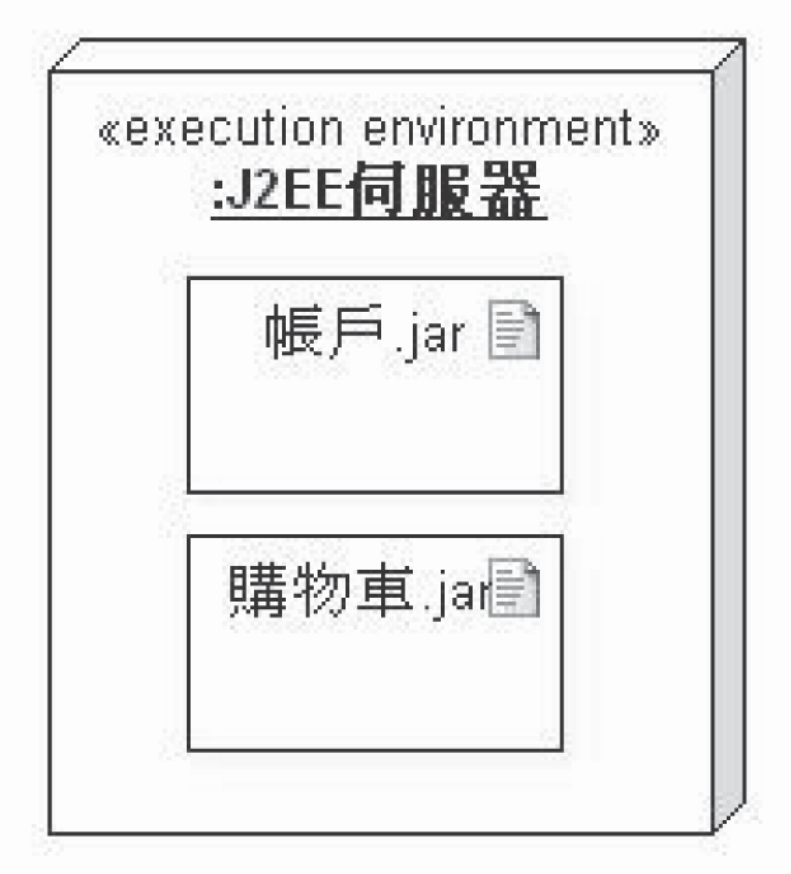


部署圖

- 節點
  - 而所謂的execution environment是指artifact的運行環境。運行環境是節點的一種，它為某些特定的元件提供了一個運行的環境；這些元件以一種可執行的artifact形式被部署於其上與執行。例如一個以J2EE技術開發的購物網站，此系統包含許多的artifacts，並且這些artifacts以jar的實體型態被部署於一個J2EE的環境中來執行，那麼我們可以將此實體結構利用部署圖塑模如圖15.16。

部署圖

- 節點

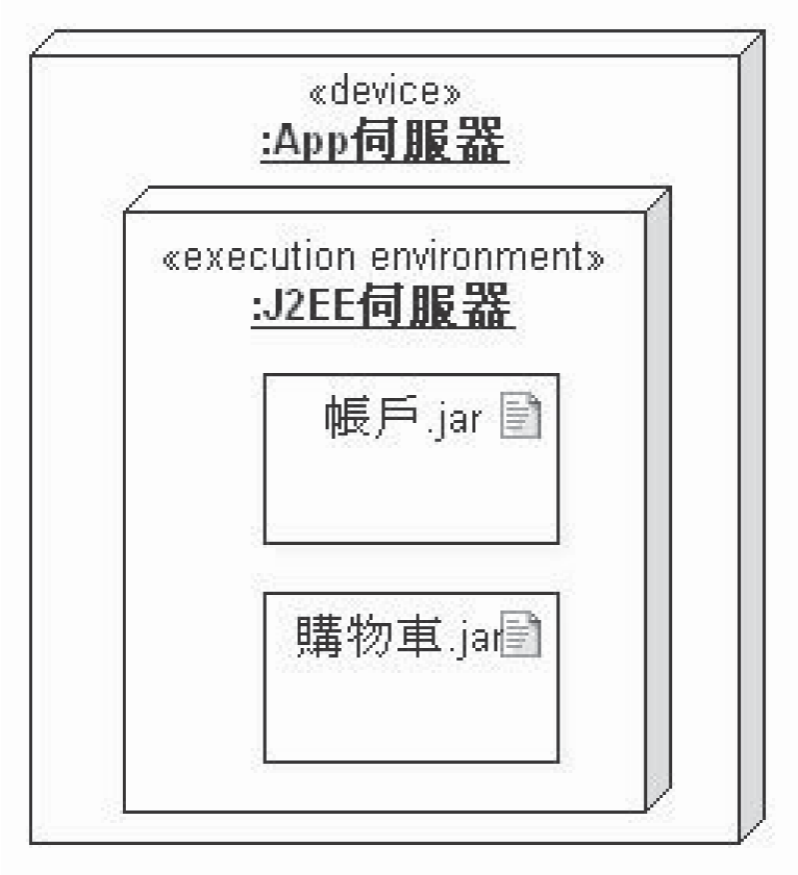


部署圖

- 節點
  - 對於節點，我們也可以利用巢狀方式來表達節點之間的組成型態。例如圖15.16的J2EE伺服器在實際上是部署於某個App伺服器中來執行的，那麼，利用巢狀方式，我們可以顯示這兩個節點的結構，如圖15.17所示。

部署圖

- 節點

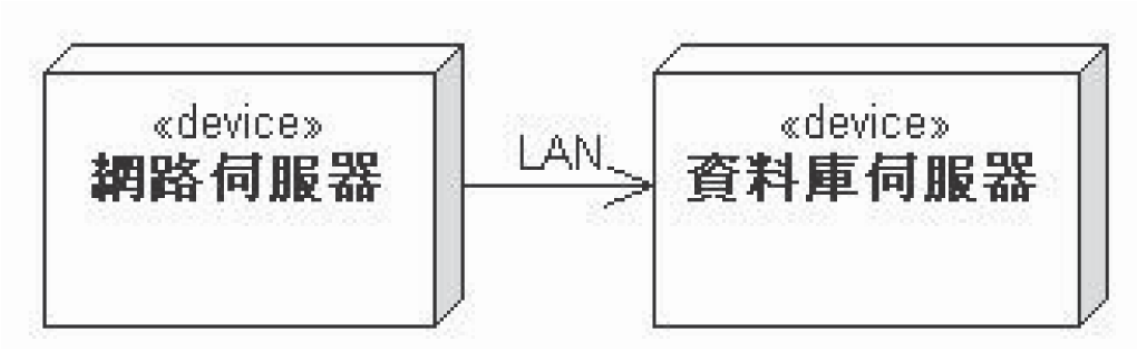

«device»
:App伺服器
«execution environment»
:J2EE伺服器
帳戶.jar
購物車.jar

部署圖

- 關係
  - 部署圖中可以包含許多的節點，節點與節點之間經由溝通管道（Communication Path）互相連接以構成網路架構；我們可以把此溝通管道想像成是資料來回的通道；溝通管道的圖形使用關聯關係來塑模。例如對於一個Web-based的網路應用程式，我們通常用資料庫來作為網站伺服器後端的資料儲存機制。如果這兩者之間的溝通管道是透過網路（LAN）來連結，那麼，此部署的情形可以繪製如圖15.18所示。

部署圖

- 關係

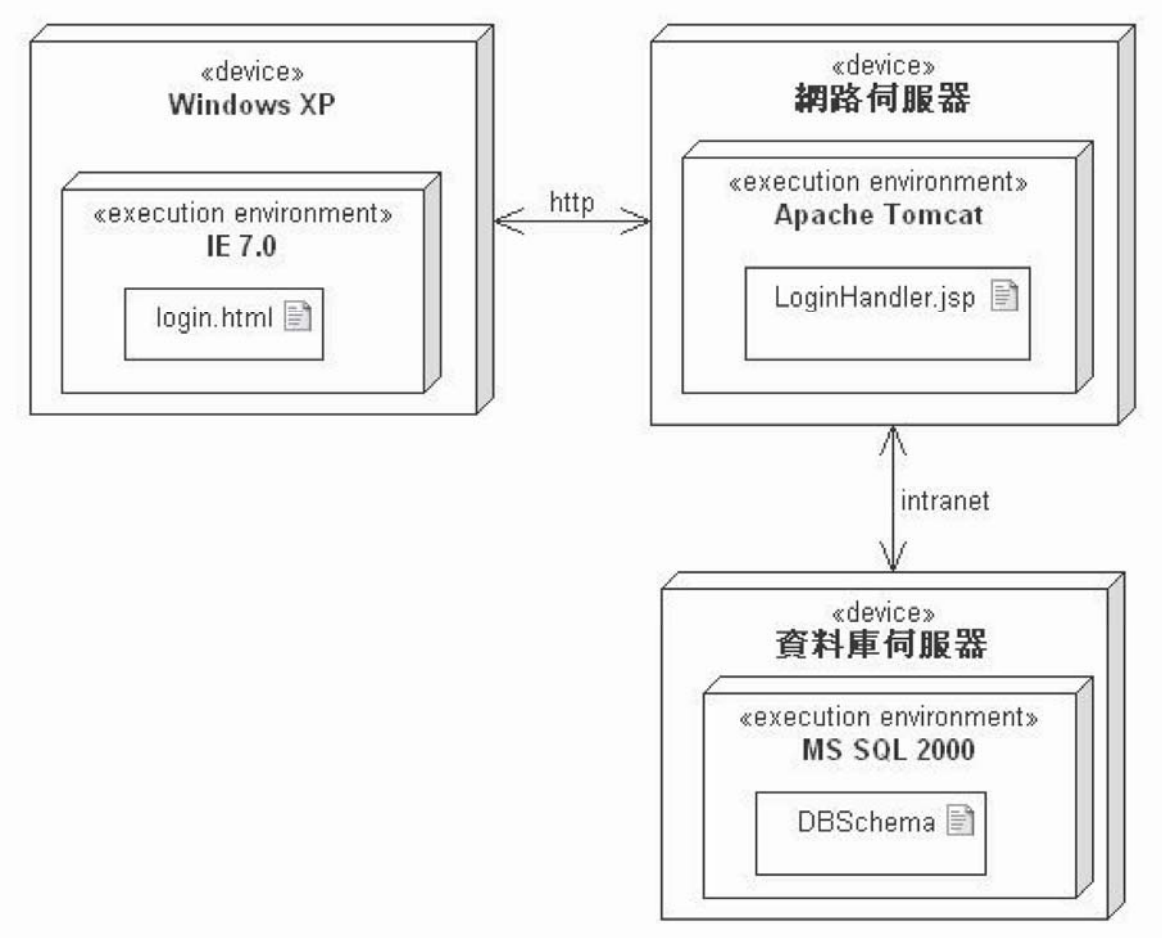


部署圖

- 關係
  - 圖15.19顯示一個較複雜的網路應用程式部署圖。此部署圖中有三個節點：Windows XP代表客戶端的機器，IE瀏覽器運行於其中，客戶端節點部署了一個HTML檔案；客戶端的節點利用HTTP作為資訊溝通的管道與遠端的網路伺服器做資料的存取溝通，伺服器中的運行環境是Apache Tomcat，其中有一個檔案LoginHandler.jsp部署於其中；網路伺服器與後端資料庫伺服器利用內部網路作為溝通管道來執行資料的存取相關動作，資料庫伺服器的運行環境是MS SQL 2000，在此節點，部署了一個檔案DBSchema。

部署圖

- 關係


«device»
Windows XP
«execution environment»
IE 7.0
login.html
http
«device»
網路伺服器
«execution environment»
Apache Tomcat
LoginHandler.jsp
intranet
«device»
資料庫伺服器
«execution environment»
MS SQL 2000
DBSchema