**JAX-WS**

Java API for XML-Based Web Services (JAX-WS) 是下一代 Web Service 程式設計模型，它補充 Java API for XML-based RPC (JAX-RPC) 程式設計模型提供的基礎。通過使用 JAX-WS，就可通過使用動態代理和 Java 注釋，借助于 Java 應用程式的更不依賴於平臺的特徵，簡化 Web Service 和客戶機的開發。

JAX-WS 是一種程式設計模型，它通過支援將基於注釋的標準模型用於開發 Web Service 應用程式和客戶機來簡化應用程式開發。JAX-WS 技術在戰略上調整了自身以適合當前朝著更以文檔為中心的消息傳遞模型發展的業界趨勢，並且替換了由 JAX-RPC 定義的遠端程序呼叫程式設計模型。雖然 JAX-RPC 程式設計模型和應用程式仍然受本產品支援，但是 JAX-RPC 具有局限性並且不支援各種複雜的以文檔為中心的服務。JAX-WS 是用於開發 Web Service 的戰略性程式設計模型，並且是 Java Platform, Enterprise Edition 6 (Java EE 6) 的必需部分。JAX-WS 也稱為 JSR 224。

**新功能部件：**

V8.0 支援 JAX-WS V2.2 和 Web Services for Java EE (JSR 109) V1.3 規範。

JAX-WS 2.2 規範將取代 JAX-WS 2.1 規範並包括此規範的功能。JAX-WS 2.2 添加了使用與 WebServiceFeature 相關的注釋（例如，@MTOM、@Addressing 和 @RespectBinding 注釋）的用戶端支持。JAX-WS 2.1 先前已在伺服器上添加對於這些注釋的支持。現在，還可以通過將 WS-Policy 斷言添加至 WSDL 文檔來實現對客戶機或服務啟用和配置 WS-Addressing 支援。此外，Web Services for Java EE 1.3 規範引入了對於這些與 WebServiceFeature 相關的注釋的支持，還支援使用部署描述符元素在客戶機和伺服器上配置這些功能。JAX-WS 2.2 要求使用 Java XML 綁定體系結構 (JAXB) V2.2 來進行資料綁定。

JAX-WS 程式設計標準的實現提供用於開發 Web Service 和客戶機的以下增強功能：

* **Java 應用程式的增強的平臺獨立性。**

通過使用 JAX-WS API，借助 Java 應用程式的增強的平臺獨立性簡化了 Web Service 和客戶機的開發。JAX-WS 利用動態代理機制來向可插入提供程式提供正式的授權模型。這是通過 JAX-RPC 獲得的增強功能，它依賴于生成特定于供應商的存根以進行調用。

* **注釋**

JAX-WS 支持對帶有中繼資料的 Java 類進行注釋，以指示 Java 類是 Web Service。JAX-WS 支援使用基於“用於 Java 程式設計語言的中繼資料設施”(JSR 175) 規範和“用於 Java 平臺的 Web Service 中繼資料”(JSR 181) 規範的注釋以及 JAX-WS 2.2 規範所定義的注釋。在 Java 原始程式碼和 Java 類中使用注釋可以簡化 Web Service 的開發。使用注釋來定義通常在部署描述符檔和 WSDL 檔中指定的資訊或用於將中繼資料從 XML 和 WSDL 檔映射到源工件中的資訊。

例如，可在 Java 原始程式碼中嵌入一個簡單的 @WebService 標記來將 Bean 顯示為 Web Service。

@WebService

public class QuoteBean implements StockQuote {

public float getQuote(String sym) { ... }

}

@WebService 注釋告訴伺服器運行時環境將該 Bean 上的所有公用方法顯示為 Web Service。其他詳細程度級別可通過在各方法或參數上添加其他注釋來控制。使用注釋可以更容易將 Java 工件顯示為 Web Service。另外，當使用一些自頂向下映射工具從 WSDL 檔創建工件時，注釋包括在原始程式碼和 Java 類中，作為與原始檔案一起捕獲中繼資料的方法。

使用注釋也可改善在團隊結構中 Web Service 的開發，因為您不需要如 JAX-RPC Web Service 所需那樣，在單個或公共部署描述符中定義每個 Web Service。將注釋與 JAX-WS Web Service 配合使用允許並行開發服務和必需的中繼資料。

For JAX-WS web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in thewebservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

例如，如果 JAX-WS Web Service 的服務實現類包含下列各項：

* + @WebService 注釋：

@WebService(wsdlLocation=”http://myhost.com/location/of/the/wsdl/ExampleService.wsdl”)

* + webservices.xml 文件指定 WSDL 文檔的其他檔案名，如下所示：
  + <webservices>
  + <webservice-description>
  + <webservice-description-name>ExampleService</webservice-description-name>
  + <wsdl-file>META-INF/wsdl/ExampleService.wsdl</wsdl-file>
  + ...
  + </webservice-description>

</webservices>

在這種情況下，部署描述符 META-INF/wsdl/ExampleService.wsdl 中指定的值覆蓋注釋值。

* **以非同步方式調用 Web Service**

借助於 JAX-WS，可以同步或非同步方式調用 Web Service。以非同步方式調用 Web Service 時，JAX-WS 添加了對輪詢和回檔機制的支援。通過使用輪詢模型，客戶機可以發出請求和取迴響應物件，對回應物件執行輪詢以確定伺服器是否已回應。當伺服器回應時，將檢索到實際的回應。通過使用回檔模型，客戶機提供回檔處理常式來接受和處理入站回應物件。輪詢模型和回檔模型都支援客戶機集中於繼續處理工作，而不等待回應返回，同時提供一個更動態和更有效的模型來調用 Web Service。

例如，Web Service 介面可包含適用於同步和非同步請求的方法。在以下示例中，非同步請求用粗體進行了標識：

@WebService

public interface CreditRatingService {

// sync operation

Score getCreditScore(Customer customer);

// async operation with polling

**Response**<**Score**> getCreditScoreAsync(Customer customer);

// async operation with callback

**Future**<**?**> getCreditScoreAsync(Customer customer,

**AsyncHandler**<**Score**> handler);

}

使用回檔機制的非同步調用需要客戶機程式師的其他輸入。回檔是一個包含應用程式碼的物件，在接收到非同步回應時將運行該代碼。使用以下代碼示例來調用非同步回檔處理常式：

CreditRatingService svc = ...;

**Future**<**?**> invocation = svc.getCreditScoreAsync(customerFred,

**new AsyncHandler**<**Score**>**() {**

**public void handleResponse (**

**Response**<**Score**> **response)**

**{**

**Score score = response.get();**

**// do work here...**

**}**

**}**

);

使用以下代碼示例來調用非同步輪詢客戶機：

CreditRatingService svc = ...;

**Response**<**Score**> **response** = svc.getCreditScoreAsync(customerFred);

while (!**response.isDone())** {

// Complete an action while we wait.

}

// No cast needed, because of generics.

Score score = **response.get()**;

* **使用資源注入**

JAX-WS 支持資源注入以進一步簡化 Web Service 的開發。JAX-WS 使用 Java EE 5 的此關鍵功能部件將在 Java 運行時環境中創建和初始化公共資源的負擔從 Web Service 應用程式轉移給應用程式容器環境本身。JAX-WS 支援 JSR-250 中為資源注入和應用程式在其運行時環境中的生命週期定義的一部分注釋。

應用程式伺服器還支援使用 @Resource 或 @WebServiceRef 注釋來聲明 JAX-WS 受管客戶機以及請求注入 JAX-WS 服務和埠。在欄位或方法中使用其中任一注釋時，該注釋會導致注入 JAX-WS 服務或埠實例。使用這些注釋還會導致將注釋所指定的類型綁定到 JNDI 名稱空間。

@Resource 注釋由包括在 Java Platform, Enterprise Edition 5 (Java EE 5) 中的 JSR-250（公共注釋）規範定義。通過將 @Resource 注釋放置在服務端點實現類中類型為javax.xml.ws.WebServiceContext 的變數上，可以請求資源注入並收集與該特定端點調用相關的 javax.xml.ws.WebServiceContext 介面。從 WebServiceContext 介面中，您可以收集請求的 MessageContext（該請求與使用 getMessageContext() 方法的特定方法調用相關聯）。

@WebServiceRef 注釋由 JAX-WS 規範進行定義。

以下示例演示將 @Resource 和 @WebServiceRef 注釋用於資源注入：

@WebService

public class MyService {

@Resource

private WebServiceContext ctx;

@Resource

private SampleService svc;

@WebServiceRef

private SamplePort port;

public String echo (String input) {

…

}

}

有關資源注入的更多資訊，請參閱 JAX-WS 規範的 5.2.1 和 5.3 節。

* **使用 JAXB 2.2 的資料綁定**

JAX-WS 利用 Java XML 綁定體系結構 (JAXB) 2.2 API 和工具作為 Java 物件與 XML 文檔之間的映射的綁定技術。JAX-WS 工具依賴 JAXB 工具對 Java 物件與 XML 文檔之間的雙向映射執行缺省資料綁定。JAXB 資料綁定替換 JAX-RPC 規範描述的資料綁定。

JAX-WS 2.2 要求使用 JAXB 2.2 進行資料綁定。JAXB 2.2 對其注釋稍作增強，從而改進了模式的生成並且能夠更好地與 JAX-WS 進行集成。

* **動態和靜態客戶機**

JAX-WS 的動態客戶機 API 稱為 Dispatch client (javax.xml.ws.Dispatch)。Dispatch client 是面向 XML 消息傳遞的客戶機。資料以 PAYLOAD 或 MESSAGE 方式發送。使用 PAYLOAD 方式時，分派客戶機只負責提供 <soap:Body> 的內容，JAX-WS 添加 <soap:Envelope> 和 <soap:Header> 元素。使用 MESSAGE 方式時，分派客戶機負責提供整個 SOAP 包絡，其中包括 <soap:Envelope>、<soap:Header> 和 <soap:Body> 元素。JAX-WS 不會將任何其他內容添加到消息中。Dispatch client 支援使用回檔或輪詢機制的非同步調用。

JAX-WS 的靜態客戶機程式設計模型稱為代理客戶機。代理客戶機根據必須提供的服務端點介面 (SEI) 調用 Web Service。

* **支持 MTOM**

通過使用 JAX-WS，您可以隨 Web Service 請求一起發送二進位附件（例如，圖像或檔）。JAX-WS 添加了對二進位資料的優化傳輸的支援，如消息傳輸優化機制 (MTOM) 指定那樣。

* **多種資料綁定技術**

JAX-WS 向最終使用者顯示以下綁定技術：XML Source、SOAP Attachments API for Java (SAAJ) 1.3 和 Java XML 綁定體系結構 (JAXB) 2.2。XML Source 使用戶能夠將javax.xml.transform.Source（它表示要處理的 Source 物件中的資料）傳遞到運行時環境。SAAJ 1.3 現在能夠通過介面傳遞整個 SOAP 文檔而不僅僅是有效內容。此操作由通過介面傳遞 SAAJ SOAPMessage 物件的客戶機完成。JAX-WS 利用 JAXB 2.2 支援作為 Java 與 XML 之間的資料綁定技術選項。

* **SOAP 1.2 支持**

已向 JAX-WS 2.0 添加對 SOAP 1.2 的支持。JAX-WS 支援 SOAP 1.1 和 SOAP 1.2，因此您可以隨 Web Service 請求一起發送二進位附件（例如，圖像或檔）。JAX-WS 添加了對二進位資料的優化傳輸的支援，如 MTOM 指定那樣。

* **開發工具**

JAX-WS 提供了 wsgen 和 wsimport 命令列工具，用於為 JAX-WS Web Service 生成可移植工件。創建 JAX-WS Web Service 時，可以從 WSDL 檔或實現 Bean 類開始。如果從實現 Bean 類開始，那麼使用 wsgen 命令列工具來生成所有 Web Service 伺服器工件，包括 WSDL 檔（如果請求的話）。如果從 WSDL 檔開始，那麼使用 wsimport 命令列工具來為伺服器或客戶機生成所有 Web Service 工件。wsimport 命令列工具處理帶有模式定義的 WSDL 檔以生成可移植工件，這些工件包括服務類、服務端點介面類別和相應 XML 模式的 JAXB 2.2 類。

* **支持 Web Services for Java EE V1.3**

Web Services for Java EE V1.3 規範支援通過使用注釋和部署描述符條目對 JAX-WS 服務和客戶機配置 MTOM、定址和 RespectBinding 功能。

* **支援對 WRAPPED 參數類型和返回類型使用空的 targetNamespace**

JAX-WS 2.2 支援方法參數和返回類型。在 JAX-WS Web Service 操作中，您可以使用指令引數和可選返回類型來定義 Web Service 操作。如果指令引數和返回類型定義了空的 targetNamespace 屬性（即，使用 @WebParam 或 @WebResult 注釋為 targetNamespace 屬性指定 "" 值），那麼 JAX-WS 運行時環境將按以下方式表現：

* + 如果操作為文檔類型、參數類型為 WRAPPED 並且參數未映射至頭，那麼會使用指令引數和返回類型來映射空的名稱空間。
  + 如果參數類型不為 WRAPPED，那麼會使用通過使用 @WebParam 或 @WebResult 注釋指定的 targetNamespace 參數的值。

# JAX-WS 客戶機程式設計模型

Java API for XML-Based Web Services (JAX-WS) Web Service 客戶機程式設計模型支援 Dispatch client API 和動態代理客戶機 API。Dispatch client API 是動態客戶機程式設計模型，而 JAX-WS 的靜態客戶機程式設計模型是動態代理客戶機。Dispatch client 和動態代理客戶機都支援同步和非同步調用 JAX-WS Web Service。

* [Dispatch client](http://www.ibm.com/support/knowledgecenter/SSAW57_8.0.0/com.ibm.websphere.nd.doc/info/ae/ae/cwbs_jaxwsclients.html?lang=zh-tw#cwbs_jaxwsclients__dispatch)：當想要在 XML 消息級別工作，或者想要在 JAX-WS 級別不使用任何生成的工件工作時，請使用此客戶機。
* [動態代理客戶機](http://www.ibm.com/support/knowledgecenter/SSAW57_8.0.0/com.ibm.websphere.nd.doc/info/ae/ae/cwbs_jaxwsclients.html?lang=zh-tw#cwbs_jaxwsclients__dynproxy)：當想要調用基於服務端點介面的 Web Service 時，請使用此客戶機。

#### Dispatch client

基於 XML 的 Web Service 將 XML 消息用於 Web Service 與 Web Service 客戶機之間的通信。JAX-WS API 提供高級方法來簡化和隱藏有關在 Java 方法調用和與它們相關聯的 XML 消息之間進行轉換的詳細資訊。然而，在某些情況下，您可能需要在 XML 消息級別工作。Dispatch client API 提供了在 XML 消息級別調用服務的支援。Dispatch client API javax.xml.ws.Dispatch 是動態 JAX-WS 客戶機程式設計介面。要編寫 Dispatch client，必須具有 Dispatch client API 和受支援的物件類型方面的專門知識，並瞭解關聯 Web 服務描述語言 (WSDL) 檔的消息表示。Dispatch client 能夠以 MESSAGE 或 PAYLOAD 方式發送資料。使用 javax.xml.ws.Service.Mode.MESSAGE 方式時，Dispatch client 負責提供整個 SOAP 包絡，其中包括<soap:Envelope>、<soap:Header> 和 <soap:Body> 元素。使用 javax.xml.ws.Service.Mode.PAYLOAD 方式時，Dispatch client 只負責提供 <soap:Body> 的內容，並且 JAX-WS 將有效內容包含在 <soap:Envelope> 元素中。

Dispatch client API 要求應用程式客戶機將消息或有效內容構造成 XML，這需要消息或消息內容的詳細知識。Dispatch client 支援下列對象類型：

* javax.xml.transform.Source：使用 Source 物件來使客戶機能夠直接使用 XML API。可以將 Source 物件與 SOAP 或 HTTP 綁定配合使用。
* JAXB 對象：借助 JAXB 物件，客戶機可以通過根據 XML 模式生成的 JAXB 物件，使用 JAX-WS 應用程式來創建和控制 XML。JAXB 物件只能與 SOAP 或 HTTP 綁定配合使用。
* javax.xml.soap.SOAPMessage：使用 SOAPMessage 物件，以便客戶機可以處理 SOAP 消息。SOAPMessage 物件只能與 SOAP 綁定配合使用。
* javax.activation.DataSource：使用 DataSource 物件，以便客戶機能夠處理多用途網際網路郵件擴充協議 (MIME) 消息。DataSource 只能與 HTTP 綁定配合使用。

例如，如果輸入參數類型為 javax.xml.transform.Source，那麼對 Dispatch client API 的調用類似於如下所示代碼示例：

Dispatch**<Source>** dispatch = … create a Dispatch**<Source>**

Source request = … create a Source object

Source response = dispatch.invoke(request);

Dispatch 參數值確定 invoke() 方法的返回類型。

Dispatch client 用下列三種方法之一調用：

* 使用 invoke 方法對請求和回應同步調用
* 帶回檔或輪詢物件使用 invokeAsync 方法對請求和回應非同步調用
* 使用 invokeOneWay 方法單向調用

請參閱 JAX-WS 規範第 4 章第 3 節，以獲取有關如何使用 Dispatch client 的更多資訊。

#### 動態代理客戶機

JAX-WS 的靜態客戶機程式設計模型稱為動態代理客戶機。動態代理客戶機根據必須提供的服務端點介面 (SEI) 調用 Web Service。動態代理客戶機類似於 Java API for XML-based RPC (JAX-RPC) 程式設計模型中的存根客戶機。雖然 JAX-WS 動態代理客戶機與 JAX-RPC 存根客戶機都基於從 WSDL 檔生成的服務端點介面 (SEI)，但仍存在一個主要差別。動態代理客戶機在運行時使用 Java 5 動態代理功能動態生成，而基於 JAX-RPC 的存根客戶機是由工具生成的不可移植的 Java 文件。與 JAX-RPC 存根客戶機不同，動態代理客戶機不要求您在應用程式伺服器上運行客戶機之前針對不同的供應商重新生成存根，因為生成的介面不需要特定供應商資訊。

動態代理實例擴展 java.lang.reflect.Proxy 類並利用基本 Java SE 運行時環境 (JRE) 6 中的動態代理功能。然後，客戶機應用程式可以提供用於創建代理實例的介面，而運行時負責動態創建表示 SEI 的 Java 物件。

動態代理客戶機用下列三種方法之一調用：

* 使用 invoke 方法對請求和回應同步調用
* 帶回檔或輪詢物件使用 invokeAsync 方法對請求和回應非同步調用
* 使用 invokeOneWay 方法單向調用

有關使用動態代理客戶機的更多資訊，請參閱 JAX-WS 規範的第 4 章。

**JAX-WS 應用程式封裝**

可以將用於 XML Web Service (JAX-WS) 應用程式的 Java 應用程式設計發展介面 (API) 封裝成 Web Service。JAX-WS Web Service 包含在 Web 應用程式歸檔 (WAR) 檔中或企業歸檔 (EAR) 檔的 WAR 模組中。

支援 JAX-WS 的 WAR 檔包括：

* WEB-INF/web.xml 文件
* 帶注釋的類，用於實現應用程式模組中包含的 Web Service
* [可選] 描述應用程式模組中包含的 Web Service 的 Web Service 描述語言 (WSDL) 文檔。

WEB-INF/web.xml 檔類似於以下示例：

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_ID" xmlns=”http://java.sun.com/xml/ns/j2ee”

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"

version="2.4">

</web-app>

web.xml 可包含 Servlet 或 Servlet 映射元素。如果不需要對 web.xml 檔進行定制，那麼會在裝入模組時，WebSphere® Application Server 運行時會動態定義它們。有關配置 web.xml 檔的更多資訊，請閱讀有關在 JAX-WS 應用程式的 web.xml 檔中定制 Web URL 模式的資訊。

帶注釋的類必須至少包含一個包括 @WebService 注釋的 Web Service 實現類。與 Web Service 相關的注釋的定義和規範由 JAX-WS 和 JSR-181 規範提供。Web Service 實現類可以存在於WEB-INF/classes 目錄中，也可以存在於 WAR 檔的 WEB-INF/lib 目錄中包含的 Java 歸檔 (JAR) 檔中。

可以選擇是否在 JAX-WS 應用程式封裝中包括 WSDL 文檔。如果省略特定 Web Service 的 WSDL 文檔，那麼 WebSphere Application Server 運行時將根據 Web Service 實現類中包含的注釋動態構造 WSDL 定義。