## J2EE Components

Le applicazioni J2EE sono fatte da componenti. Un *componente J2EE* è un’unità software funzionale auto-contenuta assemblata in un’applicazione J2EE con le relative classi e files e che comunica con gli altri componenti:

* Applicazione client e applets sono componenti eseguiti sul client.
* Le tecnologie Java Servlet e pagine JavaServer™ (JSP™) sono componenti web eseguite sul server.
* I componenti Enterprise JavaBeans™ (enterprise beans) sono componenti di business eseguiti sul server.

I componenti J2EE sono scritti in linguaggio Java e sono compilati nello stesso modo di qualsiasi altro programma in questo linguaggio. La differenza tra i componenti J2EE e le classi Java “standard” è che i componenti J2EE sono assemblati in un’applicazione J2EE, viene verificato che siano ben formate e conformi con le specifiche J2EE, e sono utilizzate per la produzione, dove sono eseguite e controllate dal server J2EE.

## J2EE Clients

Un client J2EE può essere un client web o un’applicazione client.

### Client Web

Un *client web* consiste in due parti: (1) una pagina web dinamica contenente vari tipi di linguaggio di markup (HTML, XML, e così via), generate da componenti web eseguiti a livello web, e (2) un browser web, che restituisce le pagine ricevute dal server.

Un client web è qualcosa chiamato *thin client*. I thin clients di solito non interrogano i database, eseguono complesse regole di business o si connettono ad applicazioni obsolete. Quando usi un thin client, tali operazioni pesanti sono scaricate agli enterprise beans eseguiti sul server J2EE, dove possono fare leva su sicurezza, velocità, servizi, e affidabilità delle tecnologie del lato server J2EE.

### Applets

Una pagina web ricevuta dal livello web può includere un’applet incorporata. Un’*applet* è una piccola applicazione client scritta in linguaggio Java che viene eseguita in una virtual machine installata nel browser web. Tuttavia, i sistemi client necessiteranno probabilmente del Plug-in Java e possibilmente un file di sicurezza affinché l’applet si esegua in modo corretto sul browser.

I componenti web sono le API migliori per creare un programma client web poiché non necessitano di plug-in o file di sicurezza sul sistema client. Inoltre, i componenti web consentono un design dell’applicazione più pulito e componibile poiché forniscono un modo per separare la programmazione delle applicazioni dal design della pagina web. Chi sviluppa il design della pagina non necessita così di capire a sintassi del linguaggio Java.

### Application Client

Un’*applicazione client* viene eseguita sulla macchina client e fornisce un modo agli utenti di occuparsi dei compiti che richiedono un’interfaccia utente più ricca che può essere fornita da un linguaggio di markup. Generalmente ha un’interfaccia grafica utente (GUI) creata da Swing o dall’Abstract Window Toolkit (AWT) API, ma un’interfaccia a command-line è sicuramente possibile.

Le applicazioni client accedono direttamente agli enterprise beans a livello business. Tuttavia, se i requisiti dell’applicazione lo autorizzano, un’applicazione client può aprire una connessione HTTP per stabilire una connessione con un server eseguito a livello web.

### The JavaBeans™ Component Architecture

I livelli server e client possono inoltre includere componenti basati sull’architettura del componente JavaBeans per gestire il flusso di dati tra l’applicazione client o applet e i componenti eseguiti sul server J2EE, o tra i componenti server e un database. I componenti JavaBeans non sono considerati componenti J2EE dalle specifiche J2EE.

I componenti JavaBeans hanno proprietà e metodi get e set per accedere alle proprietà. I componenti JavaBeans usati in questo modo sono generalmente semplici per design e implementazione ma dovrebbero essere conformi per denominazione e convenzione di design sottolineati nell’architettura dei componenti JavaBeans.

### J2EE Server Communications

La figura mostra i vari elementi possono comporre il livello client. Il client comunica con il livello business eseguito sul server J2EE direttamente o, come nel caso di un client eseguito in un browser, attraverso pagine JSP o servlet eseguite a livello web.

La tua applicazione J2EE usa un thin client basato sul browser o una densa applicazione client. Per decidere quale utilizzare, dovresti essere consapevole dello scambio tra la conservazione della funzionalità sul client e vicinanza all’utente (client denso) e lo scaricamento di più funzionalità possibile al server (thin client). Più funzionalità scarichi al server, più facile è da distribuire, schierare, e gestire l’applicazione: tuttavia, mantenere più funzionalità sul client può generare un’esperienza migliore per l’utente.

### Web Components

I componenti web J2EE sono servlets o pagine create utilizzando la tecnologia JSP (pagine JSP). Le *Servlets* sono classi Java che processano dinamicamente le richieste e le risposte del costruttore. Le *pagine JSP* sono documenti di testo che funzionano come le servlets ma consentono un approccio più improntato a creare un contenuto statico.

Le pagine HTML statiche e le applets sono impacchettate con componenti web durante l’assemblaggio ma non sono considerate componenti web dalle specifiche J2EE. Le classi utility del lato server possono essere inoltre impacchettate con le componenti web e, come le pagine HTML, non sono considerate componenti web.

Come mostra la figura, il livello web, come il livello client, può includere un componente JavaBeans per gestire l’input dell’utente e inviare l’output agli enterprise beans eseguiti a livello business per essere processati.

### Business Components

Il codice di business, ovvero la logica che risolve o incontra le necessità di un particolare dominio di business come banking, vendita al dettaglio o finanza, è gestito da enterprise beans eseguiti a livello business. La figura mostra come un enterprise bean riceve dati da un programma client, li processa (se necessario), e li invia all’enterprise information system per essere memorizzati. Un enterprise bean inoltre richiama i dati dalla memoria, li processa (se necessario), e li restituisce al programma client.

Ci sono tre tipi di enterprise beans: session beans, entity beans e message-driven beans. Un *session bean* rappresenta una conversazione transitoria con il client. Quando il client finisce l’esecuzione, il session bean e i suoi dati sono cancellati. All’opposto, un *entity bean* rappresenta dei dati persistenti memorizzati in una riga di una tabella di un database. Se il client termina o se il server cessa l’attività, il servizio sottostante assicura che l’entity bean sia salvato. Un *message-driven bean* combina le caratteristiche di un session bean e di un ricevitore messaggi Java Message Service (JMS), consentendo ad un componente business di ricevere messaggi JMS asincronicamente.

### Enterprise Information System Tier

L’ enterprise Information System gestisce il software EIS e include l’enterprise infrastructure system così come l’enterprise resource planning (ERP), il mainframe di processo delle transazioni, i sistemi database, e altri sistemi di informazioni obsolete. Per esempio, i componenti di un’applicazione J2EE può necessitare l’accesso all’enterprise information system per la connettività del database.

## J2EE Containers

Di norma, le applicazioni multilivello thin client sono difficili da scrivere poiché includono molte righe di codice intricato per gestire le transazioni e la gestione dello stato, il multithreading, il raggruppamento delle risorse e altri dettagli complessi a basso livello. L’architettura basata sui componenti e indipendente dalla piattaforma di J2EE rende le applicazioni J2EE facili da scrivere poiché la logica di business è organizzata in componenti riutilizzabili. In aggiunta, il server J2EE fornisce servizi sottostanti in forma di contenitore per ogni tipo di componente. Perciò non hai bisogno di sviluppare questi servizi da solo, sei libero di concentrarti sulla risoluzione di problemi di business.

### Containers Services

I *Containers* sono interfacce tra un componente e una funzionalità a basso livello specifica della piattaforma che supporta il componente. Prima che un componente web, un enterprise bean, o un componente dell’applicazione client possa essere eseguito, dev’essere assemblato in un modulo J2EE e dislocato nel proprio container.

Il processo di assemblaggio coinvolge la specificazione di importazioni del container per ogni componente dell’applicazione J2EE stessa. Le impostazioni del container personalizzano il supporto sottostante fornito dal server J2EE, includendo i servizi come la sicurezza, la gestione delle transazioni, le ricerche Java Naming and Directory Interface™ (JNDI) , e la connettività remota. Di seguito alcuni punti fondamentali:

* Il modello della sicurezza J2EE permette di configurare un componente web o un enterprise bean così che le risorse del sistema siano accessibili solo da utenti autorizzati.
* Il modello della transazione J2EE permette di specificare le relazioni tra metodi che generano una singola transazione così che tutti i metodi in una transazione siano trattati come una singola unità.
* I servizi di ricerca JNDI forniscono un’interfaccia unificata per denominazioni e servizi di directory multipli così che i componenti dell’applicazione possono accedere ai servizi di denominazione e directory.
* Il modello della connettività remota J2EE gestisce le comunicazioni a basso livello tra i clients e gli enterprise beans. Dopo che un enterprise bean è stato creato, un client invoca metodi su questo come se fosse nella stessa virtual machine.

Poiché l’architettura J2EE fornisce servizi configurabili, i componenti dell’applicazione possono comportarsi in maniera differente in base a dove sono dislocate. Per esempio, un enterprise bean può avere delle impostazioni di sicurezza che gli consentono un sicuro livello di accesso ai dati di un database in un ambiente di produzione e un altro livello di accesso al database in un altro ambiente di produzione.

Il container gestisce inoltre i servizi non configurabili così come il ciclo di vita degli enterprise beans e delle servlets, il raggruppamento delle risorse di connessione al database, la persistenza dei dati e l’accesso alle piattaforme API J2EE. Benché la persistenza dei dati è un servizio non configurabile, l’architettura J2EE permette di scavalcare (override) la persistenza della gestione del container includendo il codice appropriato nell’implementazione dell’enterprise bean quando desideri più controllo della persistenza di default fornita dalla gestione del container. Per esempio, potresti usare la persistenza della gestione di un bean per implementare il tuo metodo di ricerca o creare una cache di database personalizzata.

### Container Types

Il processo di dislocamento installa le componenti dell’applicazione J2EE nel container J2EE illustrato in figura.



#### J2EE Server

La porzione di esecuzione del prodotto J2EE. Un server J2EE fornisce EJB e container web.

#### Enterprise JavaBeans (EJB) Container

Gestisce l’esecuzione degli enterprise beans per le applicazioni J2EE. Gli enterprise beans e i loro contenitori sono eseguiti sul server J2EE.

#### Web Container

Gestisce l’esecuzione dei componenti della pagina JSP e della servlet per le applicazioni J2EE. I componenti web e i loro contenitori sono eseguiti sul server J2EE.

#### Applicazione Client Container

Gestisce l’esecuzione dei componenti dell’applicazione client. Le applicazioni clients e i loro container sono eseguiti sul client.

#### Applet Container

Gestisce l’esecuzione delle applets. Consiste in un web browser e un Java Plug-in eseguiti sul client insieme.

## Web Services Support

I servizi web sono applicazioni enterprise basate sul web che usano protocolli standard e di trasporto, aperti, basati su XML, per scambiare dati con i clients che li richiamano. La piattaforma J2EE fornisce le API degli XML e gli strumenti necessari per progettare, sviluppare, testare, e dislocare i servizi web e i clients che cooperano completamente con gli altri servizi web e clients eseguiti su piattaforme Java o non Java.

Per scrivere servizi web e clients con le API XML J2EE, tutto ciò di cui hai bisogno è di passare i dati dei parametri alla chiamata del metodo e processare i dati restituiti; o per i servizi web orientati ai documenti, inviare i documenti contenenti i dati di servizio avanti e indietro. Non è necessario programmare a basso livello poiché le implementazioni API XML fanno il lavoro di tradurre i dati dell’applicazione da e a un flusso di dati in XML trasferito dal protocollo di trasporto standardizzato in XML. Questi standard e protocolli XML sono introdotti nella prossima sezione.

La traduzione di dati in un flusso di dati standardizzati in XML è ciò che rende i servizi web e i clients scritti con le API XML J2EE completamente cooperanti.

Ciò non significa necessariamente che i dati trasportati includano tags XML poiché i dati trasportati possono essi stessi essere semplice testo, dati XML, o qualsiasi tipo di dati binari come audio, video, mappe, files di programma, documenti Computer-aided Design (CAD) e così via. La prossima sezione introduce gli XML e spiega come le parti che si occupano del business possono usare i tags XML e gli schemi per scambiare dati in modo significativo.

### XML

L’XML è uno standard di testo, estensibile, multipiattaforma, per rappresentare i dati. Quando un XML è scambiato tra le parti, queste sono libere di creare i propri tags per descrivere i dati, assemblare gli schemi per specificare quali tags possono essere usati in un particolare tipo di documento XML, e usare i fogli di stile (stylesheets) per gestire la visualizzazione e la gestione dei dati.

Per esempio, un servizio web può usare XML e uno schema per produrre liste di prezzi, e le compagnie che ricevono le liste di prezzi e gli schemi possono avere i propri fogli di stile per gestire i dati nel modo che meglio si adatta alle necessità. Di seguito degli esempi:

* Una compagnia potrebbe inserire le informazioni XML dei prezzi in un programma per tradurre XML in HTML così da poterle postare nella propria rete.
* Una compagnia partner potrebbe inserire le informazioni XML dei prezzi in uno strumento per creare una presentazione di marketing.
* Un’altra compagnia potrebbe leggere le informazione XML dei prezzi in un’applicazione per l’elaborazione.

### SOAP Transport Protocol

Le richieste del client e le risposte del servizio web sono trasmesse come messaggi Simple Object Access Protocol (SOAP) per il HTTP per abilitare uno scambio completamente cooperante tra clients e web services, tutto eseguito su diverse piattaforme e in varie zone di internet. Il HTTP è uno standard di richiesta-e-risposta conosciuto per inviare messaggi per internet, e SOAP è un protocollo XML che segue il modello richiesta-e-risposta di HTTP.

La porzione SOAP di un messaggio trasportato gestisce ciò che segue:

* Definisce una “busta” in XML per descrivere cosa c’è nel messaggio e come processare il messaggio.
* Include le regole di codifica XML per esprimere le istanze dei tipi di dati definiti nell’applicazione.
* Definisce le convenzioni in XML per rappresentare la richiesta al servizio remoto e la risposta risultante.

### Standard WSDL Format

Il Web Service Description Language (WSDL) è un formato standardizzato XML per descrivere i servizi di network. La descrizione include i nomi del servizio, la locazione del servizio, e i modi per comunicare con il servizio. Le descrizioni del servizio WSDL possono essere memorizzate in registri UDDI o pubblicate sul web (o entrambe). Il Sun Java System Application Server Platform Edition 8 fornisce uno strumento per generare la specifica WSDL di un servizio web che utilizza le chiamate della procedura remota per comunicare con i clients.

### Standard UDDI ed ebXML Formats

Altri standards XML, come lo Universal Description, Discovery and Integraton (UDDI) ed ebXML, rendono possibile per le aziende la pubblicazione di informazioni su Internet riguardo ai propri prodotti e servizi web, dove le informazioni possono essere leggibili e accessibili globalmente dai clienti che desiderano fare affari.

## Packaging Applications

Un’applicazione J2EE è distribuita in un file Enterprise Archive (EAR), un file Java Archive (JAR) standard con l’estensione .ear. Usando i files EAR e i moduli è possibile assemblare un numero di diverse applicazioni J2EE utilizzando alcuni degli stessi componenti. Non è necessario ulteriore codice: è solo questione di assemblare (o impacchettare, “packaging”) vari moduli J2EE nei files EAR J2EE.

Un file EAR (in figura) contiene moduli J2EE e deployment descriptors. Un *deployment descriptor* è un documento XML con estensione .xml che descrive le impostazioni di dislocamento di un’applicazione, un modulo, o un componente. Poiché le informazioni del deployment descriptor sono dichiarative, può essere modificata senza dover modificare il codice sorgente. Al runtime, il server J2EE legge il deployment descriptor e agisce di conseguenza sull’applicazione, sul modulo, o sul componente.

Ci sono due tipi di deployment descriptor: J2EE e runtime. Un *J2EE deployment descriptor* è definito da una specifica J2EE e può essere utilizzato per configurare le impostazioni di dislocamento su qualsiasi implementazione compatibile J2EE. Un *runtime deployment descriptor* è utilizzato per configurare dei parametri specifici di implementazione J2EE. Per esempio, il runtime deployment descriptor di Sun Java System Application Server Platform Edition 8 contiene informazioni come la context root di un’applicazione web, la mappatura dei nomi trasferibili delle risorse di un’applicazione alle risorse di un server, e parametri specifici di implementazione dell’Application Server, come le direttive di caching. Il runtime deployment descriptor dell’Application Server è denominato sun-*moduleType*.xml ed è allocato nella stessa directory del J2EE deployment descriptor.

Un *modulo J2EE* consiste uno o più componenti J2EE per lo stesso tipo di container e un componente deployment descriptor di quel tipo. Un deployment descriptor di un modulo di un enterprise bean, per esempio, dichiara gli attributi di transazione e le autorizzazioni sulla sicurezza per l’enterprise bean. Un modulo J2EE senza un deployment descriptor dell’applicazione può essere dislocata come un modulo *stand-alone.* I quattro tipi di moduli J2EE sono i seguenti:

* Moduli EJB, che contengono i files delle classi per l’enterprise bean e un deployment descriptor del EJB. I moduli EJB sono impacchettati come files JAR con l’estensione .jar.
* Moduli web, che contengono i files delle classi servlet, i files JSP, i file delle classi di supporto, i file GIF e HTML, e un deployment descriptor dell’applicazione web. I moduli web sono impacchettati in file JAR con l’estensione .war (web archive).
* Moduli application client, che contengono i files delle classi e un deployment descriptor dell’applicazione client. I moduli application client sono impacchettati come files JAR con estensione .jar.
* Moduli resource adapter, che contengono tutte le interfacce Java, le classi, le librerie native, e altra documentazione, insieme a un deployment descriptor del resource adapter. Insieme, questi implementano la Connector architecture per un particolare EIS. I moduli resource adapter sono impacchettati come file JAR con estensione .rar (resource adapter archive).

## Development Roles

I moduli riutilizzabili rendono possibile separare lo sviluppo dell’applicazione e lo sviluppo del processo in parti distinte cosicché le diverse persone o compagnie possano eseguire diverse parti del processo.

Le prime due parti coinvolgono l’acquisto e l’installazione del prodotto e degli strumenti J2EE. Dopo aver acquistato e installato il prodotto, i componenti J2EE possono essere sviluppate dai providers dei componenti dell’applicazione, assemblati dagli assemblatori dell’applicazione, e sviluppati dagli sviluppatori dell’applicazione. In una grande organizzazione, ognuna di queste parti dev’essere eseguita da diversi individui o teams. Questa divisione del lavoro funziona poiché ognuna di queste parti primitive genera un file trasferibile, che sarà un input in una parte successiva. Per esempio, in una fase di sviluppo di un componente dell’applicazione, uno sviluppatore software enterprise bean produce files EJB JAR. Nella parte di assemblaggio dell’applicazione, un altro sviluppatore combina questi files EJB JAR in un’applicazione J2EE e la salva come un file EAR. Nella parte di distribuzione dell’applicazione, un amministratore di sistema sul sito del cliente utilizza il file EAR per installare l’applicazione J2EE in un server J2EE.

Le diverse parti non sono sempre eseguite da persone diverse. Se lavori in una piccola compagnia, per esempio, o se stai creando in prototipo di un’applicazione campione, potresti eseguire i compiti in ogni fase.

### J2EE Product Provider

Il J2EE product provider è la compagnia che progetta a rende disponibile per l’acquisto le API della piattaforma J2EE, e altre caratteristiche definite nella specifica di J2EE. I product providers sono tipicamente sistemi operativi, sistemi database, server d’applicazione, o fornitori di web server che implementano la piattaforma J2EE secondo le specifiche della Java 2 Platform, Enterprise Edition.

### Tool Provider

Il tool provider è la compagnia o la persona che crea gli strumenti di sviluppo, assemblaggio e packaging usati dai component providers, assemblatori, e sviluppatori.

### Application Component Provider

L’application component provider è la compagnia o la persona che crea i componenti web, gli enterprise beans, le applets, o le applicazioni client per utilizzare le applicazioni J2EE.

#### Enterprise Bean Developer

Uno sviluppatore di enterprise bean assolve seguenti compiti per distribuire un file EJB JAR contenente gli enterprise beans:

* Scrive e compila il codice sorgente
* Specifica il deployment descriptor
* Impacchetta i file .class e il deployment descriptor nel file EJB JAR

#### Web Component Developer

Uno sviluppatore di web component assolve i seguenti compiti per distribuire un file WAR contenente il componente (o componenti) web:

* Scrive e compila il codice sorgente servlet
* Scrive i file JSP e HTML
* Specifica il deployment descriptor
* Impacchetta i files .class, .jsp e .html e il deployment descriptor in un file WAR

#### Application Client Developer

Uno sviluppatore dell’applicazione client assolve i seguenti compiti per distribuire un file JAR contenente l’applicazione client:

* Scrive e compila il codice sorgente
* Specifica il deployment descriptor per il client
* Impacchetta i files .class e il deployment descriptor in un file JAR

### Application Assembler

L’assembler dell’applicazione è la compagnia o la persona che riceve i moduli dell’applicazione dai component providers e li assembla in un file applicazione J2EE EAR. L’assembler o lo sviluppatore possono modificare il deployment descriptor direttamente o usando strumenti che aggiungono correttamente tags XML secondo le selezioni interattive. Uno sviluppatore software assolve i seguenti compiti per distribuire un file EAR contenente l’applicazione J2EE:

* Assembla i file EJB JAR e WAR creati nella fase precedente in un file applicazione J2EE (EAR)
* Specifica il deployment descriptor per l’applicazione J2EE
* Verifica che i contenuti del file EAR siano ben formati e conformi con le specifiche J2EE

### Application Deployer e Administrator

Il “diffusore” dell’applicazione e amministratore è la compagnia o la persona che configura e sviluppa l’applicazione J2EE, amministra l’infrastruttura di computing e networking dove viene eseguita l’applicazione J2EE, e supervisiona l’ambiente di runtime. I compiti includono cose come i controlli d’impostazione della transazione, gli attributi sulla sicurezza, e la specificazione delle connessione al database.

Durante la configurazione, il diffusore segue le seguenti istruzioni fornite dal component provider dell’applicazione per risolvere le dipendenze esterne, specificare le impostazioni di sicurezza, e assegnare gli attributi di transazione. Durante l’installazione, il diffusore sposta i componenti dell’applicazione al server e genera le classi e le interfacce specifiche del container.

Un diffusore o amministratore di sistema assolve i seguenti compiti per installare e configurare un’applicazione J2EE:

* Aggiunge il file applicazione J2EE (EAR) creato nella fase precedente al server J2EE
* Configura l’applicazione J2EE per l’ambiente operazionale modificando il deployment descriptor dell’applicazione J2EE
* Verifica che i contenuti del file EAR siano ben formati e conformi alle specifiche J2EE
* Rilascia (installa) il file applicazione J2EE (EAR) nel server J2EE

## J2EE 1.4 APIs

La figura illustra la disponibilità delle APIs della piattaforma J2EE 1.4 in ogni tipo di container J2EE. La seguente sezione da un breve riepilogo delle tecnologie richieste dalla piattaforma J2EE e dalle APIs enterprise di J2SE che dovrebbero essere utilizzate nell’applicazione J2EE.



### Enterprise JavaBeans Technology

Un componente Enterprise JavaBeans™ (EJB™), o *enterprise bean*, è il corpo del codice avente campi e metodi per implementare i moduli della logica di business. Puoi pensare a un enterprise bean come ad un blocco di costruzione che può essere usato da solo o con altri enterprise bean per eseguire la logica di business sul server J2EE.

Come detto in precedenza, ci sono tre tipi di enterprise beans: session beans, entity beans e message-driven beans. Gli enterprise beans spesso interagiscono con i database. Uno dei benefici degli entity beans è che non hai bisogno di scrivere codice SQL o usare il JDBC™ API direttamente per eseguire le operazioni di accesso al database: l’EJB container lo gestisce per te. Tuttavia, se superi (override) la persistenza di default controllata dal container per qualche ragione, dovrai utilizzare il JDBC API.

### Java Servlet Technology

La tecnologia delle Java servlet permette di definire classi servlet specifiche HTTP. Una classe servlet estende la capacità del server che ospita le applicazioni visitate tramite il modello richiesta-risposta. Sebbene le servlets possono rispondere ad ogni tipo di richiesta, sono comunemente usate per estendere le applicazioni ospitate dai servers web.

### JavaServer Pages Technology

La tecnologia JavaServer Pages™ (JSP™) permette di inserire frammenti di codice direttamente di un documento di testo. Una pagina JSP è un documento di testo che contiene due tipi di testo: dati statici (che possono essere espressi in qualsiasi formato di testo come HTML, WML e XML) e elementi JSP, che determinano come la pagina costruisce contenuti dinamici.

### Java Message Service API

Il Java Message Service (JMS) API è uno standard di messaggi che permette ai componenti dell’applicazione J2EE di creare, inviare, ricevere e leggere messaggi. Ciò agevola la comunicazione distribuita che è debolmente accoppiata, affidabile, e asincrona.

### Java Transaction API

La Java Transaction Api (JTA) fornisce un’interfaccia standard per demarcare le transazioni. L’architettura J2EE fornisce un auto commit di default per gestire le transazioni di commit e di rollback. Un *auto commit* significa che qualsiasi altra applicazione che sta osservando i dati vedrà i dati aggiornati dopo ogni operazione di lettura o scrittura sul database. Tuttavia, se la tua applicazione esegue due separate operazioni di accesso al database che dipendono l’una dall’altra, vorrai utilizzare la JTA API per demarcare dove l’intera transazione, incluse entrambe le operazioni, inizia, esegue il roll back, ed esegue il commit.

### JavaMail API

Le applicazioni J2EE utilizzano la JavaMail™ API per inviare notifiche via email. La JavaMail API è composta da due parti: un’interfaccia a livello dell’applicazione utilizzata dalle componenti dell’applicazione per inviare email, e un interfaccia service provider. La piattaforma J2EE include JavaMail con un servizio provider che permette alle componenti dell’applicazione di inviare Internet mail.

### JavaBeans Activation Framework

Il JavaBean Activation Framework (JAF) è incluso perché usato da JavaMail. Il JAF fornisce servizi standard per determinare il tipo di un arbitrario frammento di dati, incapsula l’accesso ad esso, scopre le operazioni disponibili su di esso, e crea i componenti JavaBeans appropriati per eseguire queste operazioni.

### Java API for XML Processing

Il Java API for XML Processing (JAXP) supporta l’elaborazione di documenti XML usando il Document Object Model (DOM), la Simple API for XML (SAX), e l’Extensible Stylesheet Language Transformations (XSLT). JAXP agevola le applicazioni nell’analisi e trasformazione di documenti XML indipendente da una particolare implementazione di elaborazione XML.

JAXP inoltre fornisce supporto allo spazio dei nomi, che permette di lavorare con schemi che potrebbero altrimenti avere conflitti tra i nomi. Progettato per essere flessibile, JAXP permette di utilizzare qualsiasi decodificatore conforme a XML o elaboratore XML dall’interno della tua applicazione, e supporta lo schema W3C. Puoi trovare informazioni sullo schema W3C al seguente URL:

<http://www.w3.org/XML/Schema>

### Java API for XML-Based RPC

La Java API for XML-Based RPC (JAX-RPC) utilizza lo standard SOAP e l’HTTP, così i programmi client possono eseguire chiamate di procedura remota (RPCs) su Internet. JAX-RPC supporta inoltre WSDL, quindi si possono importare ed esportare documenti WSDL. Con JAX-RPC e un WSDL, si può facilmente cooperare con i clients e i servizi in esecuzione su piattaforme Java e non Java come .NET. Per esempio, basato sul documento WSDL, un client Visual Basic .NET può essere configurato per utilizzare un servizio web implementato con tecnologia Java, o un servizio web può essere configurato per riconoscere un client Visual Basic .NET.

JAX-RPC si basa sul protocollo di trasporto HTTP. Facendo un passo avanti, JAX-RPC permette di creare applicazioni di servizio che combinano HTTP con una versione Java dei protocolli Secure Socket Layer (SSL) e del Transport Layer Security (TLS) per stabilire un autenticazione basica o reciproca. SSL e TSL assicurano l’integrità dei messaggi fornendo una crittografia dei dati con il client e le capacità di autenticazione del server.

### SOAP with Attachments API for Java

Il SOAP with Attachments API for Java (SAAJ) è un API a basso livello da cui dipende JAX-RPC. SAAJ agevola la produzione e il consumo di messaggi conformi alla specifica SOAP 1.1 e le note di SOAP Attachments. La maggior parte degli sviluppatori non usato SAAJ API, ma l’API a più alto livello JAX-RPC.

### Java API for XML Register

Il Java API for XML Register (JAXR) permette di accedere ai registri di business e di utilizzo generale attraverso il web. JAXR supporta gli standard ebXML Registry and Repository e l’emergente specifica UDDI. Usando JAXR, lo sviluppatore può apprendere una singola API e ottenere l’accesso a entrambe queste importanti tecnologie di registro.

Inoltre, le aziende possono inserire materiale da condividere e cercare materiale che gli altri hanno inserito. I gruppi standard hanno sviluppato schemi per tipi particolari di documenti XML: due aziende potrebbero, per esempio, acconsentire ad utilizzare lo schema come modulo di vendita standard della propria impresa. Poiché lo schema è memorizzato in un registro di business standard, entrambe le parti possono utilizzare JAXR per accedervi.

### J2EE Connector Architecture

L’architettura di J2EE Connector è utilizzata dai venditori di strumenti J2EE e dai system integrators per creare resource adapters che supportano l’accesso a sistemi enterprise di informazioni, che possono essere collegate a qualsiasi prodotto J2EE. Un *resource adapter* è un componente software che consente ai componenti dell’applicazione J2EE di accedere e interagire con il sottostante resource manager di EIS. Poiché un resource adapter è specifico del proprio resource manager, tipicamente c’è un differente resource adapter per ogni tipo di database o sistema enterprise di informazioni.

L’architettura di J2EE Connector fornisce inoltre un’integrazione transazionale orientata alle prestazioni, sicura, scalabile e improntata sui messaggi, di un servizio web J2EE con gli EISs esistenti che può essere sincrona o asincrona. Le applicazioni esistenti e gli EISs integrati attraverso l’architettura di J2EE Connector nella piattaforma J2EE può essere esposta come un servizio web XML usando i modelli JAX-RPC e del componente J2EE. Quindi JAX-RPC e l’architettura di J2EE Connector sono tecnologie complementari per l’integrazione delle applicazioni enterprise (EAI) e l’integrazione del business end-to-end.

### JDBC API

Il JDBC API permette di invocare comandi SQL da metodi Java. Si usa il JDBC API in un enterprise bean quando si fa l’override della persistenza gestita da un container di default o si ha un session bean che accede ad un database. Con la persistenza gestita dal container, le operazioni di accesso al database sono gestite dal container, e l’implementazione dell’enterprise bean non contiene codice JDBC o comandi SQL. Si può inoltre utilizzare il JDBC API da una servlet o da una pagina JSP per accedere al database direttamente senza passare attraverso un enterprise bean.

Il JDBC API è composto da due parti: un’interfaccia a livello dell’applicazione utilizzata dai componenti dell’applicazione per accedere al database, e un’interfaccia di servizio provider per collegare il driver JDBC alla piattaforma J2EE.

### Java Namnig and Directory Interface

Il Java Naming and Directory Interface™ (JNDI) fornisce la funzionalità di nomi e directory. Fornisce applicazioni con metodi per eseguire le operazioni di directory standard, come associare attributi con oggetti e la ricerca di oggetti utilizzando questi attributi. Utilizzando JNDI, un’applicazione J2EE può memorizzare e recuperare qualsiasi tipo di oggetto Java denominato.

Il servizio di denominazione J2EE fornisce clients di applicazione, enterprise beans, e componenti web per accedere ad un ambiente di denominazione JNDI. Un *naming environment* (“ambiente di denominazione”) permette di personalizzare un componente senza il bisogno di accedere o cambiare il codice sorgente del componente. Un container implementa l’ambiente del componente e lo fornisce al componente come *naming context* (“contesto di denominazione”)JNDI.

Un componente J2EE alloca il suo contesto di denominazione d’ambiente usando le interfacce JNDI. Un componente crea un oggetto javax.naming.InitialContext e cerca il contesto di denominazione d’ambiente in InitialContext sotto il nome di java:comp/env. Un ambiente di denominazione del componente è salvato direttamente nel contesto di denominazione d’ambiente o in uno qualsiasi dei suoi diretti o indiretti sotto-contesti.

Un componente J2EE può accedere agli oggetti nominati definiti dall’utente e forniti dal sistema. I nomi degli oggetti forniti dal sistema, come gli oggetti JTA UserTransaction, sono memorizzati in un contesto di denominazione d’ambiente, java:comp/env. La piattaforma J2EE permette a un componente di denominare oggetti definiti dall’utente, come enterprise beans, accessi d’ambiente, oggetti DataSource JDBC, e connessioni ai messaggi. Un oggetto dovrebbe essere nominato all’interno di un sotto-contesto dell’ambiente di denominazione a seconda del tipo dell’oggetto. Per esempio, gli enterprise beans sono denominati all’interno del sotto-contesto java:comp/env/ejb, e i riferimenti DataSource JDBC in un sotto-contesto java:comp/env/jdbc.

Poiché JNDI è indipendente da qualsiasi implementazione specifica, le applicazioni possono utilizzare JNDI per accedere a denominazioni multiple e servizi di directory, incluse le denominazioni e i servizi di directory esistenti come LDAP, NDS, DNS e NIS. Ciò permette alle applicazioni J2EE di coesistere con applicazioni e sistemi obsoleti. Per maggiori informazioni su JNDI, guarda *The JNDI Tutorial:*

<http://java.sun.com/products/jndi/tutorial/index.html>

### Java Authentication and Authorization Service

Il Java Authentication and Authorization Service (JAAS) fornisce un modo alle applicazioni J2EE per autenticare e autorizzare un utente specifico o un gruppo di utenti ad eseguirla.

JAAS è una versione Java del framework standard Pluggable Authentication Module (PAM), che estende l’architettura di sicurezza di Java 2 Platform per supportare le autorizzazioni per gli utenti.

### Simplified System Integration

La piattaforma J2EE è una soluzione di integrazione per tutti i sistemi, indipendente dalla piattaforma che crea un mercato aperto in cui ogni venditore può vendere a qualsiasi cliente. Tale mercato incoraggia i venditori a competere, non cercando di bloccare i clienti nelle proprie tecnologie, ma anzi cercando di superare chiunque altro nel fornire prodotti e servizi che avvantaggiano i clienti, così come una migliore prestazione, migliori strumenti, o un migliore supporto ai clienti.

Le APIs J2EE agevolano l’integrazione di sistemi e applicazioni attraverso:

* Un modello di applicazione unificato attraverso livelli con enterprise beans
* Un meccanismo di richiesta-e-risposta semplificato con pagine JSP e servlets
* Un modello di sicurezza affidabile con JAAS
* Integrazione di scambio dati XML con JAXP, SAAJ e JAX-RPC
* Interoperabilità semplificata con l’architettura di J2EE Connector
* Semplice connettività con il JDBC API
* Integrazione di applicazioni enterprise con message-driven beans e JMS, JTA e JNDI

Puoi imparare di più riguardo alla piattaforma J2EE per costruire sistemi di business integrati leggendo *J2EE Technology in Practice*, by Rich Cattell e Jim Inscore (Addison-Wesley, 2001):

[http://java.sun.com/j2ee/inpractice/aboutthebook.html](http://java.sun.com/j2ee/inpractice/aboutthebook.html%20)

## Sun Java System Application Server Platform Edition 8

Sun Java System Application Server Platform Edition 8 è un’implementazione completamente conforme della piattaforma J2EE 1.4. In aggiunta per il supporto di tutte le APIs descritte nella sezione precedente, l’Application Server include un numero di tecnologie e strumenti J2EE che non fanno parte della piattaforma 1.4 ma sono forniti come comodità per lo sviluppatore.

Questa sezione riepiloga brevemente le tecnologie e gli strumenti che compongono l’Application Server, e le istruzioni per far partire e fermare l’Application Server, far partire l’Admin Console, far partire deploytool, e far partire e fermare il PointBase database server. Gli altri capitoli spiegano come utilizzare i rimanenti strumenti.

### Technologies

L’Application Server include due tecnologie di interfaccia utente, JavaServer Pages Standard Tag Library e JavaServer™ Faces, che sono state costruite e utilizzate in congiunzione alle tecnologie della piattaforma J2EE 1.4 Java servlet e Java Server Pages.

#### JavaServer Pages Standard Tag Library

La JavaServer Pages Standard Tag Library (JSTL) incapsula il nucleo delle funzionalità comuni a molte applicazioni JSP. Anziché mischiare tags da numerosi venditori nella tua applicazione JSP, impiega un singolo set standard di tags. Questa standardizzazione permette di dislocare la tua applicazione su qualsiasi JSP container che supporta JSTL e rende più probabile che l’implementazione dei tags sia ottimizzata.

JSTL ha un iteratore e tags condizionali per gestire il controllo del flusso, tags per manipolare documenti XML tags per l’internazionalizzazione, tags per accedere ai database usando SQL, e altre funzioni comunemente usate.

#### JavaServer Faces

La tecnologia JavaServer Faces è un framework di interfaccia utente per la costruzione di applicazioni web. I componenti principale della tecnologia JavaServer Faces sono i seguenti:

* Un framework componente GUI
* Un modello flessibile per rendere i componenti in tipi differenti di HTML o diversi linguaggi e tecnologie di markup. Un oggetto Render genera il markup per rendere il componente e convertire i dati memorizzati (in un oggetto modello) in un tipo che può essere rappresentato in una view.
* Un RendeerKit standard per generare markup HTML/4.01

Le seguenti caratteristiche supportano i componenti GUI:

* Validazione input
* Gestione Eventi
* Conversione di dati tra oggetti modello e componenti
* Creazione di oggetti modello gestiti
* Configurazione di navigazione della pagina