**La Portabilità delle Applicazioni Java**

Il problema iniziale riguarda la necessità di rendere le applicazioni portabili, cioè eseguibili su qualsiasi macchina senza modificare il codice. Per risolverlo, Java sfrutta il concetto di "scrivi una volta, esegui ovunque" (Write Once, Run Anywhere), grazie alla Java Virtual Machine (JVM).

Tuttavia, un'ulteriore sfida è stata rendere queste applicazioni servibili tramite client tradizionali come i browser. Qui entrano in gioco le tecnologie basate su **application server**, che consentono di eseguire applicazioni Java all'interno di un ambiente controllato, detto **servlet container**. Un application server non è altro che una JVM avanzata, capace di ospitare e gestire molteplici applicazioni Java.

Questi server intercettano le richieste HTTP dei client e determinano quale classe Java e quali metodi invocare. Ad ogni applicazione dinamica sul server (Dynamic Server Application) corrisponde almeno una **servlet**, ovvero una classe Java conforme a determinate specifiche. L'application server, inoltre, si occupa di gestire l'intero ciclo di vita delle richieste e delle applicazioni.

**Comunicazione tra Applicazioni Java**

Un altro problema da risolvere era far comunicare due applicazioni Java remote, che potevano trovarsi su macchine diverse o su JVM diverse.

1. **Comunicazione su una stessa macchina**  
   Una soluzione semplice potrebbe essere lo scambio di file serializzati. Tuttavia, questo metodo è efficace solo se le applicazioni girano sulla stessa macchina.
2. **Comunicazione tra JVM su macchine diverse**  
   Per questo scopo sono state sviluppate tecnologie come gli **Enterprise JavaBeans (EJB)**. Gli EJB rappresentano una soluzione standardizzata per consentire a due applicazioni Java di comunicare, anche a distanza, attraverso una chiamata remota. Questa tecnica crea un **segnaposto** nella JVM locale, che punta a un metodo situato in una JVM remota.

**Tipi di EJB**

Gli Enterprise JavaBeans sono suddivisi in tre categorie principali, ognuna progettata per risolvere esigenze specifiche:

1. **Message-Driven Beans (MDB)**  
   Gli MDB sono orientati alla programmazione asincrona e vengono veicolati tramite messaggi. La loro filosofia è semplice: consentono a più client di sottoscriversi a un servizio e ricevere i risultati quando disponibili. Questo approccio è ideale per scenari in cui il client può proseguire la propria esecuzione senza attendere immediatamente il completamento di una richiesta.

Un esempio tipico è un servizio con una **coda** di messaggi, dove i client richiedono lo stesso servizio in modo asincrono. Gli MDB si occupano di notificare i sottoscrittori quando i risultati sono pronti.

1. **Entity Beans**  
   Gli Entity Bean rappresentano un'interfaccia per accedere ai dati e implementano le classiche operazioni di CRUD (Create, Read, Update, Delete). In passato erano molto usati, ma oggi il loro ruolo è stato sostituito da tecnologie più moderne come i DAO (Data Access Object).
2. **Session Beans**  
   I Session Bean sono il tipo di EJB più comune e vengono utilizzati per stabilire una connessione tra client e server con un ciclo di vita definito. Offrono uno **stato conversazionale**, cioè una connessione che può mantenere informazioni specifiche tra una richiesta e l'altra. Si suddividono in tre tipologie:
   * **Stateful**: Mantengono una connessione stabile tra client e server. Quando un client interagisce con un oggetto stateful, il server si "ricorda" di quel client, garantendo che tutte le richieste successive siano gestite dalla stessa istanza. Questo è utile per applicazioni come carrelli di e-commerce, dove è necessario mantenere uno stato specifico durante la sessione.
   * **Stateless**: Non mantengono uno stato conversazionale. Ogni richiesta è indipendente e non c'è garanzia che un client riceva la stessa istanza di un bean per richieste successive. Gli stateless sono ideali per operazioni che non richiedono un contesto persistente.
   * **Singleton**: Viene creata un’unica istanza condivisa da tutti i client. Questa istanza è ideale per scenari in cui si desidera condividere risorse o configurazioni comuni tra tutti i client.

**Ciclo di Vita di un EJB**

Gli EJB seguono un ciclo di vita gestito dal server:

1. **Creazione**: Il bean viene creato nel container del server, che invoca il costruttore della classe.
2. **Iniezione delle dipendenze**: Il server verifica se il bean necessita di oggetti esterni per funzionare e li fornisce.
3. **Post-Creazione**: Vengono eseguiti metodi speciali di inizializzazione.
4. **Ready State**: Il bean è pronto per essere utilizzato. Il server mantiene un pool di bean per ottimizzare le prestazioni.
5. **Passive State**: Il bean viene messo in una modalità inattiva quando non è più utilizzato attivamente, ma può essere ripristinato successivamente.
6. **Distruzione**: Quando il bean non è più necessario, il server lo rimuove dalla memoria, eseguendo eventuali metodi di pulizia, come la chiusura delle connessioni al database.

**Inversione del Controllo (IoC)**

Un altro concetto fondamentale per lo sviluppo Java è l’**Inversione del Controllo (IoC)**. Questa tecnica consente di configurare dinamicamente l’istanziazione di un oggetto al di fuori del codice Java stesso. Normalmente, quando un oggetto viene creato nel codice, è necessario conoscere i costruttori e i metodi da chiamare. Ciò crea una forte dipendenza tra l’oggetto chiamante e l’oggetto chiamato.

Con l’IoC, queste informazioni vengono spostate all’esterno del codice, riducendo la rigidità e migliorando la manutenibilità del software. Un framework che implementa IoC è Spring, che utilizza la **Dependency Injection** per fornire automaticamente gli oggetti necessari a un bean. Questo approccio semplifica la gestione delle dipendenze e consente di scrivere codice più modulare ed estensibile.

**La Portabilità delle Applicazioni Java**

La portabilità delle applicazioni Java è un concetto fondamentale che consente di eseguire il codice su diverse piattaforme senza modifiche. Questo obiettivo è raggiunto grazie alla **Java Virtual Machine (JVM)**, che funge da intermediario tra il codice Java e l'hardware sottostante, permettendo l'esecuzione di applicazioni Java su qualsiasi sistema operativo che supporti la JVM.

**Ruolo della JVM**

La JVM interpreta il bytecode Java, generato dal compilatore Java, e lo esegue sulla macchina locale. Questo approccio consente di mantenere la compatibilità del codice su diverse piattaforme hardware e software. La frase "scrivi una volta, esegui ovunque" riassume perfettamente questo principio, rendendo Java una scelta popolare per lo sviluppo di applicazioni distribuite.

**Application Server e Servlet**

Per rendere le applicazioni Java accessibili tramite client tradizionali come i browser, sono stati introdotti gli **application server**. Questi server forniscono un ambiente controllato per l'esecuzione delle applicazioni Java, gestendo le richieste **HTTP** e determinando quale servlet invocare. Un **servlet** è una classe Java che risponde a richieste web, elaborando dati e restituendo contenuti dinamici.*Un application server non è altro che una JVM avanzata*, capace di ospitare più applicazioni Java e gestire il loro ciclo di vita. Ogni applicazione dinamica sul server corrisponde a una servlet, che gestisce le interazioni con i client.

**Comunicazione tra Applicazioni Java**

La comunicazione tra applicazioni Java su macchine diverse o su JVM diverse è un'altra sfida affrontata dalla tecnologia Java. Ci sono due scenari principali:

1. **Comunicazione su una stessa macchina**: In questo caso, le applicazioni possono scambiarsi dati tramite file serializzati.
2. **Comunicazione tra JVM su macchine diverse**: Qui entrano in gioco gli **Enterprise JavaBeans (EJB)**. Gli EJB forniscono un modo standardizzato per consentire la comunicazione remota tra applicazioni Java, creando un segnaposto locale (stub) nella ***JVM locale*** che punta a un metodo situato in una ***JVM remota.***

**Tipi di EJB**

Gli Enterprise JavaBeans sono suddivisi in tre categorie principali:

1. **Message-Driven Beans (MDB)**:
   * Gli MDB sono progettati per gestire la programmazione asincrona attraverso messaggi. Permettono ai client di sottoscriversi a servizi e ricevere notifiche quando i risultati sono disponibili.
   * **Esempio pratico**: Un servizio di notifica che invia aggiornamenti agli utenti quando ci sono nuovi messaggi in coda.
2. **Entity Beans**:
   * Rappresentano un'interfaccia per l'accesso ai dati e implementano operazioni CRUD (Create, Read, Update, Delete). Sebbene in passato fossero molto utilizzati, oggi il loro ruolo è stato in gran parte sostituito da tecnologie più moderne come la **Java Persistence API (JPA)**.
   * **Esempio pratico**: Un bean che rappresenta un record in un database, consentendo operazioni di salvataggio e recupero dei dati.
3. **Session Beans**:
   * Sono utilizzati per stabilire una connessione tra client e server con un ciclo di vita definito.   
     ***Si dividono in tre tipi:***
     + **Stateful**: Mantengono uno stato conversazionale tra client e server.
       - **Esempio pratico**: Un carrello della spesa che memorizza gli articoli aggiunti dall'utente durante la sessione.
     + **Stateless**: Non mantengono uno stato conversazionale; ogni richiesta è indipendente.
       - **Esempio pratico**: Un servizio di calcolo che restituisce risultati senza necessità di mantenere informazioni tra le chiamate.
     + **Singleton**: Viene creata un'unica istanza condivisa da tutti i client.
       - **Esempio pratico**: Un bean che gestisce configurazioni globali per l'applicazione.

**Ciclo di Vita di un EJB**

Il ciclo di vita degli EJB è gestito dal ***container EJB*** e comprende diverse fasi:

1. **Creazione**: Il bean viene creato nel container del server.
2. **Iniezione delle dipendenze**: Il server fornisce oggetti esterni necessari al funzionamento del bean.
3. **Post-Creazione**: Vengono eseguiti metodi speciali di inizializzazione.
4. **Ready State**: Il bean è pronto per essere utilizzato.
5. **Passive State**: Il bean viene messo in modalità inattiva quando non è più utilizzato attivamente.
6. **Distruzione**: Quando non è più necessario, il bean viene rimosso dalla memoria.

**Inversione del Controllo (IoC)**

L'Inversione del Controllo (IoC) è un principio fondamentale nello sviluppo software che consente di gestire le dipendenze tra oggetti in modo più flessibile. Invece di creare direttamente le istanze degli oggetti nel codice, IoC delega questa responsabilità a un framework esterno.

Benefici dell'IoC

* Riduce le dipendenze dirette tra componenti, migliorando la manutenibilità del codice.
* Favorisce la scrittura di codice più modulare ed estensibile.

**Ruolo di Framework come Spring**

Il framework Spring implementa l'IoC attraverso la **Dependency Injection**, fornendo automaticamente gli oggetti necessari a un bean al momento della creazione. Questo approccio semplifica notevolmente la gestione delle dipendenze e consente agli sviluppatori di concentrarsi sulla logica aziendale piuttosto che sulla configurazione tecnica.In sintesi, la portabilità delle applicazioni Java è garantita dalla JVM e supportata da tecnologie come gli application server e gli EJB, che facilitano la comunicazione remota e la gestione della logica aziendale in modo standardizzato ed efficiente.