ANALISI ARCHITETTURALE SPRING FRAMEWORK

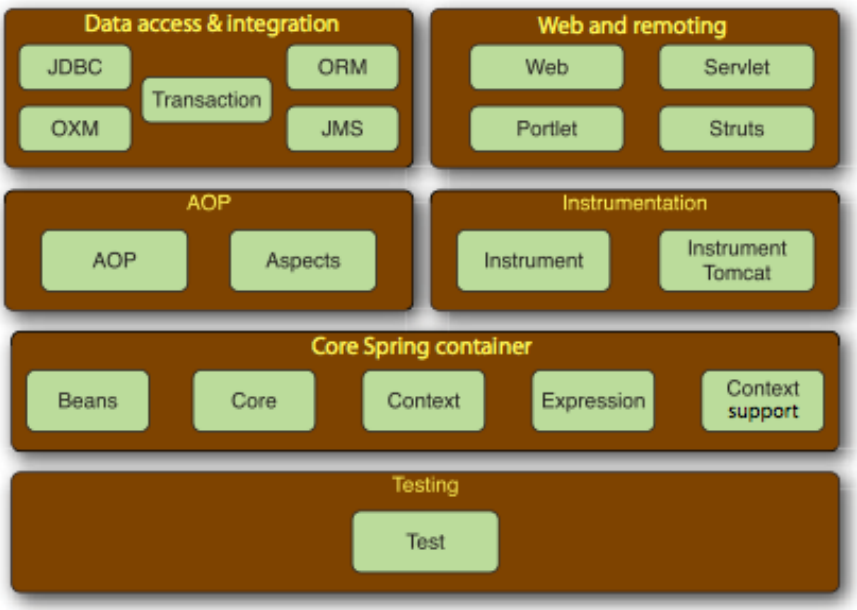
Spring è un framework open-source creato da Rod Johnson e descritto nel libro Expert One-on-One: J2EE Design and Development. Spring è stato creato per affrontare lo sviluppo di applicazioni enterprise mediante plain-old Java objects (POJOs) senza la complessità degli Enterprise Java Beans (EJB). Tale framework può essere usato con successo per qualsiasi applicazione Java e non solo per sviluppare sistemi lato-server. Milioni di sviluppatori nel mondo usano il Framework Spring per creare codice avente le seguenti caratteristiche: alte performance, facilmente testabile e riusabile. Per semplificare lo sviluppo di sistemi in Java, Spring utilizza quattro principi fondamentali:

- Utilizzo di Plain Old Java Objects (POJO) che rendono l'attività di sviluppo leggera e minimale
- Basso accoppiamento attraverso un uso intenso di interfacce e iniezione delle dipendenze
- Programmazione dichiarativa attraverso programmazione orientata agli aspetti
- Uso di templates per evitare di riscrivere sempre stesse porzioni di codice.

Spring Framework è composto da diversi moduli. Scaricando e decomprimendo una distribuzione di Spring si trovano 20 file JAR. Ogni JAR appartiene ad una delle sei seguenti categorie:

- Testing;
- Core Spring Container;
- AOP;
- Instrumentation;
- Dara Access & Integration;
- Web and Remoting.

La seguente figura schematizza l'architettura fondamentale di Spring.



Core Spring Container

Il "Core Spring Container" è il cuore di Spring. E' un contenitore che gestisce il ciclo di vita di oggetti di qualsiasi natura che, in Spring, vengono detti beans. Il suo compito è quindi quello di creare, configurare e gestire tali beans. All'interno di questo modulo si trova la "Spring Bean Factory" che è la porzione di Spring che permette l'iniezione delle dipendenze. Sfruttando il lavoro di questa Factory è possibile effettuare l'iniezione delle dipendenze e la configurazione di Spring in diversi modi. Più in dettaglio **i moduli di Core e Beans** sono responsabili delle funzionalità di *Inversion Of Control (IoC)* e *Dependency Injection*, **Il modulo context estende i servizi basilari del Core** aggiungendo le funzionalità tipiche di un moderno framework. Tra queste troviamo JNDI, EJB, JMX, internazionalizzazione(I18N) e supporto agli eventi. Infine il **modulo Expression** fornisce un potente linguaggio per interrogare e modificare oggetti a runtime.

AOP

Il modulo denominato AOP fornisce il supporto alla programmazione orientata agli aspetti. Come l'iniezione delle dipendeze, l'AOP permette di avere un basso accoppiamento tra gli oggetti che costituiscono l'applicazione. In Spring l'utilizzo di AOP offre il meglio di sé nella **gestione delle transazioni**, permettendo di evitare l'utilizzo degli EJB per tale scopo.

Data Access & Integration

Questo modulo si occupa di due aspetti fondamentali: accesso ai dati e integrazione. Tale porzione dell'ecosistema Spring fornisce un livello di astrazione per l'accesso ai dati mediante tecnologie eterogenee tra loro come ad esempio JDBC, Hibernate o JDO. Questo modulo tende a nascondere la complessità delle API di accesso ai dati, semplificando ed uniformando quelle che sono le problematiche legate alla gestione delle connessioni, delle transazioni e delle eccezioni. Notevole attenzione è stata posta sull'integrazione del framework con i principali ORM in circolazione compresi JPA, JDO, Hibernate, e iBatis. Il modulo Integration fornisce un'astrazione per usare in modo più efficiente la tecnologia JMS – Java Message Service per l'integrazione asincrona con altre applicazioni. Spring offre anche svariate implementazioni di Object/XML Mapper come JAXB, Castor e XMLBean. Più in dettaglio:

- **JDBC**: e' un modulo che fornisce un livello di astrazione a JDBC che elimina la necessità di implementare a mano tutto il codice necessario a JDBC.
- **ORM**: modulo che fornisce un livello di integrazione con alcune popolari API che implementano il mapping tra oggetti Java e tabelle di un database relazionali. Tali API includono il supporto a JPA, JDO, Hibernat ed iBatis.
- **OXM**: modulo che fornisce un livello di astrazione che supporta il mapping tra oggetti Java e documenti XML, fornisce implementazioni di JAXB, Castor, XMLBeans, JiBX ed XStream.
- **JMS**: modulo che fornisce il Java Messaging Service, contiene funzionalità per produrre e consumare messaggi.
- **Transaction**: modulo che fornisce un supporto per la gestione delle transazioni dichiarative per classi che implementano interfacce speciali e per tutti i POJO.

Web and Remoting

Essendo il Model-View-Controller (MVC) un paradigma comunemente usato per sviluppare applicazioni web, Spring è basato su tale modello. Il mondo Java è ricco di framework MVC. Apache Struts, JSF, WebWork e Tapestry sono i più famosi. Il punto di forza di Spring è che permette anche l'integrazione con questi framework. Entrando nel dettaglio:

- **Web**: modulo che fornisce le funzioni di integrazione base orientate al web, come ad esempio la funzione multipart file-upload e l'inizializzazione dello IoC container facente uso di servlet listeners ed un application context orientato al web.
- **Web-Servlet**: modulo che contiene l'implementazione del pattern Model-View-Contoller (MVC) per sviluppare Web Application.
- **Web-Struts**: modulo che contiene le classi di supporto per integrare un'applicazione Struts all'interno di un'applicazione Spring.
- **Web-Portlet**: modulo che fornisce un'implementazione MVC da usare nello sviluppo di una web application basata su portlet. Tale modulo ripropone in tale modello di sviluppo le funzionalità del modulo Web-Servlet.

Tale porzione dell'ecosistema Spring, oltre alle funzionalità per costruire applicazioni web, fornisce diverse opzioni per la comunicazione remota come Remote Method Invocation (RMI), Hessian, Burlap e JAX-WS.

Testing

Riconoscendo l'importanza del testing nello sviluppo del software, Spring mette a disposizione un modulo dedicato il cui compito è quello di testare le applicazioni sviluppate con Spring. Questo livello mette a disposizione un ambiente molto potente per il test di componenti Spring, grazie anche alla sua integrazione con JUnit e TestNG e alla presenza di Mock objects per il testing del codice in isolamento.

Quanto appena illustrato costituisce solo una minima parte delle potenzialità offerte da Spring. In realtà esistono altri framework e librerie che fanno parte del mondo di Spring che ne incrementano il suo valore. Di seguito una rapida rassegna di altri moduli che estendono le funzionalità basilari di Spring.

Spring web Flow: sfrutta le funzionalità di Spring MVC per permettere la realizzazione di applicazioni web flow-based

Spring Web Services: permette di sviluppare web services in modo più rapido ed efficiente

Spring Security: utilizzando l'AOP permette di definire, in modo dichiarativo, aspetti relativi alla sicurezza di una applicazione

Spring Integration: offre l'implementazione di diversi patterns di integrazione

Spring Batch: permette di eseguire operazioni di movimentazione massiva di dati in modo automatico, schedulato e fail-safe

Spring Social: permette di sviluppare e integrare applicazioni all'interno di social network quali Facebook e Twitter

Spring Mobile: sostiene e facilita lo sviluppo di applicazioni web mobili

Spring DM - Dynamic Modules: utilizzando Spring DM è possibile costruire applicazioni che sono formate da diversi moduli, debolmente accoppiati e fortemente coesi che espongono e consumano servizi all'interno del framework OSGi

Spring LDAP: fornisce un modello di accesso a LDAP

Spring Rich Client: toolkit che permette di sviluppare applicazioni Swing

Spring.NET: offre le caratteristiche di basso accoppiamento attraverso iniezione delle dipendenze e programmazione orientata agli aspetti sulla piattaforma .NET

Spring-Flex: permette di far interagire applicazioni Flex e AIR con i beans di Spring mediante BlazeDS.

Spring Roo: fornisce degli strumenti interattivi per permettere uno sviluppo rapido di applicazioni Java EE appoggiate allo Spring Framework

Spring Extensions: altre estensioni fornite da sviluppatori di terze parti che offrono nuove funzionalità.

Iniezione delle dipendenze

Il cuore di Spring è basato sul principio dell'iniezione delle dipende (Dependency Injection - DI) o inversione di controllo (Inversion of Control - IoC). Usando l'iniezione delle dipendenze il codice che ne scaturirà sarà semplice, facile da capire e testare. Qualsiasi applicazione non banale è costituita da due o più classi che collaborano l'una con l'altra per realizzare una logica di business. Tradizionalmente ogni oggetto ha il compito di ottenere i riferimenti degli oggetti con cui collabora. Questo comporta un codice fortemente accoppiato e difficile da testare.

Prendiamo in esame la seguente classe

```
public class Cavaliere {

    private Spada spada;

    public Cavaliere(){
        spada = new Spada();
    }

}
```

Ogni qualvolta sarà istanziato un Cavaliere questo creerà la sua spada. In questo modo è stata creata una dipendenza tra la classe Cavaliere e Spada. Utilizzando l'iniezione delle dipende il codice del precedente esempio può essere così modificato:

```
public class Cavaliere {

    private Spada spada;

    public Cavaliere(Spada spada){
        this.spada = spada;
    }

}
```

In questo secondo caso la classe Cavaliere non si deve preoccupare della creazione della Spada. La classe Spada verrà creata da una factory ed implementata in maniera totalmente indipendente e verrà fornita alla classe Cavaliere nel momento in cui essa verrà istanziata. L'intera proceduta verrà mediata dal Framework Spring. Nell'esempio appena visto è stato totalmente rimosso il controllo dalla classe Cavaliere ed è stato riposto altrove (ad esempio in un file di configurazione XML, in una classe di configurazione di Spring o mediante annotazioni all'interno del codice). La dipendenza (nell'esempio la classe Spada) sarà creata da una factory (come tutti i bean facenti parte di un'applicazione Spring) e sarà iniettata all'interno della classe che la usa (ovvero Cavaliere) tramite uno strumento chiamato Class Constructor. In questo modo il controllo del flusso è stato invertito dalla Dependency Injection poiché la gestione delle dipendenze è stata delegata ad un sistema esterno. Ora la classe Cavaliere non è accoppiata a nessuna implementazione di Spada. Nel secondo frammento di codice mostrato è molto più facile usare metodi di testing. Come si può capire dall'esempio precedente i vantaggi nell'uso dell'iniezione delle dipendenze sono molteplici e possono essere riassunti cosi:

- **Codice Collante Ridotto**: la DI permette di scrivere meno codice per far colloquiare i diversi componenti di una applicazione. In alcuni casi il codice è banale e consiste, come nell'esempio visto, semplicemente nella creazione di una nuova istanza. In altri casi le righe di codice da scrivere possono essere piuttosto complesse. Le maggiori criticità si possono verificare quando si devono effettuare delle ricerche in repository JNDI oppure quando si devono invocare risorse remote. In tali casi l'adozione della DI semplifica il codice dal momento che può fornire un servizio automatico di lookup del JNDI ed il proxy automatico di risorse remote.
- **Configurazione dell'applicazione semplificata**: Adottando l'iniezione delle dipendenze il processo di configurazione di una applicazione è semplificato. E' infatti possibile usare le annotazioni oppure file XML per specificare quali risorse devono essere iniettate in una classe specifica. Inoltre in alcuni casi l'uso della DI permette di cambiare, in modo semplice ed efficace, l'implementazione di una dipendenza. Per esempio si consideri il caso in cui un DAO (Data Access Obiect) esegue delle operazioni su un database PostgreSQL. Se, per qualche motivo, il database deve cambiare è sufficiente cambiare l'iniezione della dipendenza senza apportare modifiche dirette al codice.
- **Gestire dipendenze comuni in un singolo repository**. Usando la DI tutte le informazioni sulle dipendenze sono localizzate in un unico repository (file XML o apposita classe Java) rendendo il processo sulla gestione delle dipendenze più semplice e meno soggetto ad errori.
- **Attività di testing più efficiente**. Progettando le classi per l'iniezione delle dipendenze è molto facile, come visto, cambiare le dipendenze. Ciò è particolarmente utile nell'attività di test. Si consideri un oggetto di business che effettua alcune elaborazioni complesse. Per eseguire tali compiti usa un DAO per accedere ai dati memorizzati in un database relazionale. In fase di test non interessa verificare le azioni svolte dal DAO ma è di interesse andare a controllare il comportamento dell'oggetto di business fornendogli diversi insieme di dati. In un approccio tradizionale, dove un oggetto di business è responsabile di ottenere

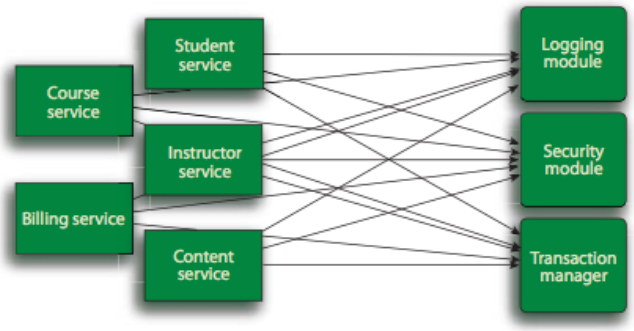
una istanza del DAO, si hanno dei problemi in fase di testing perché non si riesce a sostituire l'implementazione del DAO con un mock che restituisca l'insieme dei dati di test. Usando l'iniezione delle dipendenze è possibile creare un mock del DAO che restituisca l'insieme dei dati di test da passare all'oggetto di business per i controlli del caso. Questo meccanismo può essere esteso per testare ogni livello dell'applicazione e questo è particolarmente utile per verificare i componenti web in cui è possibile creare mock di HttpServletRequest e HttpServletResponse.

Programmazione Orientata agli Aspetti

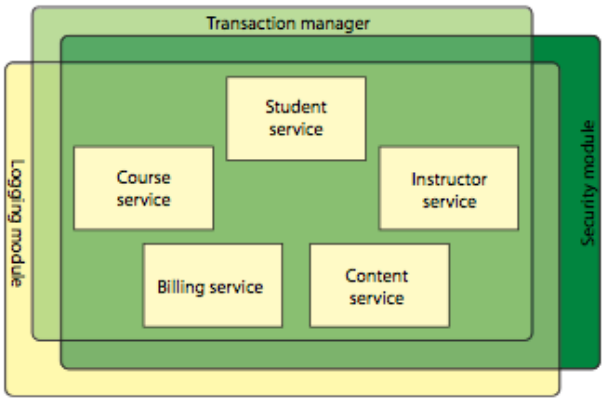
Sebbene l'iniezione delle dipendenze permetta di ottenere un basso accoppiamento, la programmazione orientata agli aspetti (AOP) permette di catturare funzionalità che sono utilizzate in tutta l'applicazione in componenti riusabili. L'AOP è definita come una tecnica che favorisce la separazione degli interessi all'interno di un sistema software. Un generico sistema è formato da diversi componenti, ciascuno dei quali è responsabile di una specifica funzionalità. Caratteristiche di un sistema come logging, gestione delle transazioni e sicurezza spesso vengono affidate a componenti le cui responsabilità fondamentali sono altre. Tali caratteristiche, siccome sono dislocate in più componenti, sono chiamate cross-cutting concerns. Allocando tali funzioni su più componenti si introducono due livelli di complessità nel codice:

- Il codice che implementa tali funzionalità è sparso su più componenti. Questo significa che se bisogna effettuare un cambiamento in una delle funzionalità è possibile che le modifiche abbiamo ripercussioni su più componenti. Anche se una singola funzionalità è stata allocata in un modulo separato resta il problema delle chiamate di metodo che sono duplicate in più punti.
- I componenti sono pieni di codice che non implementa le funzionalità di base per le quali il componente è stato pensato. Un metodo per aggiungere una voce di rubrica dovrebbe preoccuparsi solo di svolgere questo compito e non occuparsi di aspetti come sicurezza o transazionalità.

La seguente figura mostra tale complessità.



Gli oggetti di business sulla sinistra dell'immagine sono troppo legati con i servizi di sistema. Ogni componente, come mostra la figura, deve svolgere i compiti di base per i quali è stato progettato e deve occuparsi del logging, della sicurezza e di essere in un contesto transazionale. AOP consente di modularizzare tali servizi per renderli fruibili, in modo dichiarativo, ai componenti che ne fanno richiesta. Questo permette pertanto di avere dei componenti più coesi che si occupano solo di svolgere compiti per i quali sono stati progettati.



Come si vede dalla figura soprastante, servizi come la sicurezza, logging e transazioni sono utili a tutti i componenti dell'applicazione e, usando una metafora, possono essere paragonati ad una coperta che deve avvolgere tutta l'applicazione. I componenti interni all'applicazione come Student Service, Course Service si devono occupare solo della logica di business. L'applicazione tuttavia ha bisogno di servizi quali sicurezza, logging e transazioni e tali servizi, grazie all'AOP fornita da Spring, possono essere usati in modo flessibile e trasparente.

Semplificare l'integrazione con JEE

Negli ultimi anni i framework che fanno uso dell'iniezione delle dipendenze come Spring hanno guadagnato un largo consenso all'interno della comunità degli sviluppatori. Sempre un maggior numero di programmatori sceglie di usare framework come Spring ignorando un approccio basato su EJB. Per facilitare il lavoro degli sviluppatori sono state introdotte le versioni 3.0, 3.1 e 3.2 degli EJB che hanno semplificato l'API della specifica degli Enterprise Java Beans abbracciando inoltre molti dei concetti dell'iniezione delle dipendenze. Tuttavia, tutte le applicazioni che sono state costruite usando gli EJB o per sistemi realizzati con Spring per i quali si ha la necessità di rilasciarli su container JEE per utilizzare servizi tipici di piattaforme enterprise come JTA Transaction Manager, data source connection pooling, JMS connection factories Spring fornisce un supporto semplificato. Per gli EJB, Spring mette a disposizione un metodo per eseguire il lookup nel registry JNDI ed iniettare il riferimento dell'EJB all'interno dei beans di Spring. E' possibile usare Spring per fare l'azione opposta ovvero iniettare all'interno dell'EJB i beans di Spring.

Il "**Java Naming and Directory Interface**" (JNDI) è una API Java che consente di ricercare in una directory degli oggetti in base al nome. JNDI è generalmente usato nelle applicazioni JAVA EE per effettuare operazioni come:

- Transazioni (UserTransaction & TransactionManager)
- Pool di connessioni transazionali ai database (XADatasource)
- Connettersi a JMS e utilizzare queue e/o topic
- Connettersi ad un EJB
- Sicurezza

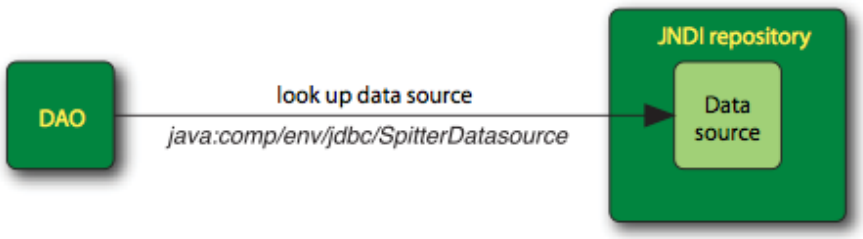
Per tutte le risorse referenziate da un registry JNDI, Spring elimina la necessità di scrivere del codice complesso per effettuare il lookup. Come conseguenza avremmo che l'applicazione sarà disaccoppiata dal JNDI e questo permette di avere del codice riusabile. Usando l'API convenzionale di JNDI il codice che solitamente è scritto assomiglia al seguente:

```
InitialContext ctx = null;
try{

    ctx = new InitialContext();
    DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/SpitterDatasource");

}catch(NamingException ne){
    //handle naming exception ...
}finally{
    if(ctx != null){
        try{
            ctx.close();
        } catch(NamingException ne) {}
    }
}
```

Dal listato appena mostrato è possibile fare le seguenti osservazioni:

- Per recuperare un DataSource è necessario creare, e poi chiudere, un contesto iniziale (InitialContext). Dal punto di vista del codice non è un'aggiunta di notevoli dimensioni ma tuttavia rappresenta del codice che non è in linea con l'obiettivo di recuperare un date source.
- Bisogna gestire un'eccezione del tipo javax.naming.NamingException.
- Il frammento di codice precedente è strettamente accoppiato con un JNDI Lookup. In generale si dovrebbe recuperare un DataSource cercando di non essere dipendenti da come questo DataSource venga recuperato.
- Il codice dipende anche da uno specifico nome di JNDI (in questo caso java:comp/enc/jdbc/SpitterDatasource – che è quello usato per la ricerca nel registry JNDI). Il nome potrebbe essere esternalizzato in un file ma, in questo modo, nel listato precedente bisognerebbe inserire il codice per eseguire la ricerca nel registry JNDI a partire dalle informazioni presenti nel file di configurazione.

La seguente immagine riassume i punti appena trattati mostrando come un generico elemento DAO sia dipendente ed accoppiato al JNDI.



Per eliminare tutte le problematiche appena descritte è possibile usare Spring ed in particolare il meccanismo dell'iniezione delle dipendenze. Spring mette a disposizione il namespace **jee** dove è definito l'elemento **<jee:jndi-lookup>** che permette di eseguire in modo semplice ed efficace l'uso di un oggetto registrato nel JNDI all'interno di Spring.

Ad esempio per iniettare in Spring un EJB sono sufficienti le seguenti linee di codice all'interno del file di configurazione di Spring.

```
<jee:jndi-lookup id="sqrtBean" jndi-name="SqrtSession"/>
<bean id="sqrtService" class="client.SqrtBean">
        <property name="service" ref="sqrtBean">
</bean>
```

Il codice appena mostrato inietta nel bean client.SqrtBean l'EJB di nome SqrtSession. In questo modo nel bean di Spring chiamato sqrtService non c'è nessuna dipendenza specifica dall'EJB. Se in futuro si dovesse usare un altro bean è sufficiente cambiare l'elemento <jee:jndi-lookup> .

Alternative a Spring

Sul mercato, trovare un framework con le stesse funzionalità di Spring è complesso. Google-Guice è una libreria Java che sfrutta tutti i benefici dell'iniezione delle dipendenze. Per quanto riguarda la parte di sviluppo web i concorrenti a Spring sono molteplici. I più noti sono Struts, Tapestry, JavaServer Faces (JSF), Apache Wicket, VRaptor, SiteMesh, Xwork, WebWork.

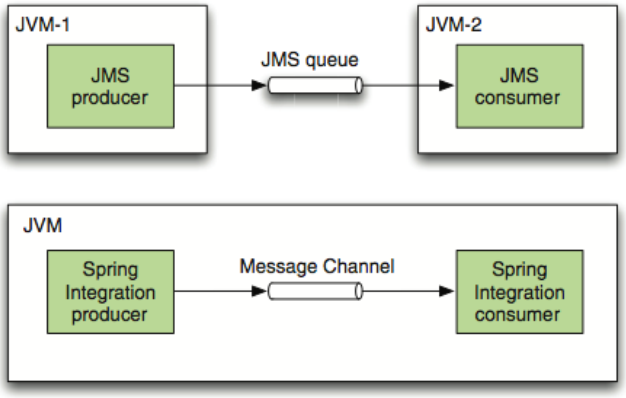
JMS

Molti sviluppatori Java, quando sentono parlare di "messaging", la prima cosa che pensano è **Java Message Service (JMS)**. JMS è l'API predominante, appartenente a Java EE, che consente ad applicazioni Java presenti in rete di scambiarsi messaggi tra loro. La specifica JMS è stata progettata per fornire un'astrazione generale sul middleware orientato ai messaggi (MOM). Esistono un gran numero di implementazioni open source della specifica JMS una delle quali è ActiveMQ. Tale prodotto, estremamente facile da configurare, è stato utilizzato negli esempi proposti nella tesi per spiegare come sia possibile fruire del messaging in Spring.

La relazione tra JMS e Spring Integration

Spring permette di usare in modo semplice il paradigma di comunicazione basato sul "Messaging". Per ottemperare a tale obiettivo viene in ausilio il modulo Spring Integration. Tale parte dell'ecosistema di Spring fornisce un modello coerente per il messaging intraprocesso e interprocesso. Quando si discute sul ruolo di JMS spesso si pensa all' integrazione di sistemi diversi ma in realtà può essere vantaggioso usare JMS anche all'interno della stessa applicazione per avere benefici quali persistenza, transazioni, bilanciamento di carico e failover. Spring Integration permette di usare facilemente JMS sia per la comunicazione tra processi dislocati su macchine diverse sia per far colloquiare processi sullo stesso calcolatore. Il ruolo primario di un channel adapter e di un messaging gateways è di connettere una destinazione locale a dei sistemi esterni senza incidere sul codice dei componenti consumatori o produttori. Un altro vantaggio che gli adattatori forniscono è la separazione tra gli aspetti relativi al messagging dai protocolli di trasporto. Quest'ultimi consentono di abilitare un messaging di tipo document-style se la particolare implementazione dell'adattatore invia le richiesta su http, interagendo con un filesystem o la mappatura verso un'altra API di messagging. Gli adattatori ed i gateways basati su JMS cadono in quest'ultima categoria e pertanto sono delle ottime scelte quando è richiesta

l'integrazione con un sistema esterno. Siccome JMS e Spring seguono lo stesso paradigma di messaging è possibile schematizzare la comunicazione intraprocesso e interprocesso usando un modello simile.



La prima parte della figura mostra la comunicazione interprocesso usando JMS mentre la seconda parte della figura mostra l'integrazione intraprocesso usando Spring Integration. Quale tipo di integrazione è più appropriato dipende dal particolare tipo di architettura dell'applicazione da realizzare. Indipendentemente dal tipo di comunicazione che si vuole utilizzare, Spring Integration fornisce una serie di qualità:

- **Load Balancing**: lo scenario è quello in cui più consumatori sono registrati presso una destinazione condivisa. Il carico può essere distribuito sulla base delle capacità degli stessi consumatori. Ad esempio può accadere che alcuni processi siano in esecuzione su macchine lente o che il processamento di alcuni messaggi necessiti di risorse ingenti e pertanto possono essere gli stessi consumatori a richiedere messaggi quando possono effettivamente elaborarli senza costringere il dispatcher a prendere decisioni.
- **Scalabilità**: I produttori potrebbero inviare tantissimi messaggi ad un unico consumatore. Per evitare un accumulo ed un ritardo nel trattamento dei messaggi, è possibile aggiungere dei processi che consumino tali messaggi nei vincoli di tempo previsti.
- **Disponibilità**: L'intero sistema deve restare operativo anche nel caso di fallimento di uno o più processi consumatori. Nel caso estremo di fallimento di tutti i processi consumatori, il broker deve essere in grado di memorizzare tutti i messaggi fino a quel momento inviati sino al ripristino di uno dei consumatori. Allo stesso modo i processi produttori possono essere allocati o deallocati senza procurare interferenze con i processi consumatori. Tale beneficio è la conseguenza del basso accoppiamento fornito da un sistema di messaggistica. Questa caratteristica può essere usata in sistemi che non devono avere un downtime. Se, per esempio, è necessario aggiornare i processi consumatori questi possono essere rimossi uno alla volta senza inficiare sulle caratteristiche di disponibilità del sistema.
- **Transazionalità**: è possibile fornire un supporto transazionale ai messaggi. Se un componente consumatore tratta con un successo un messaggio può effettuare il commit della transazione altrimenti deve fare il rollback. Quando questa caratteristica è attivata si può avere bilanciamento di carico ed inoltre, se un processo fallisce, avviene il rollback ed il messaggio può essere inviato ad un altro processo consumatore.

I benefici appena elencati sono ottenuti grazie all'uso combinato di JMS e Spring Integration. Ad esempio Spring Integration 2.1 ha il supporto per RabbitMQ, che implementa il protocollo AMQP.

SPRING e WEB SERVICE

Spring Web Service (Spring-WS) è un modulo di Spring che consente di facilitare la creazione di Web Services. In particolar modo tale libreria aiuta nella creazione di Web Services di tipo Soap con approccio "Contract-First". Usando Spring lo sviluppatore non dovrà codificare esplicitamente il WSDL che sarà invece generato dal framework a partire dall'XML della response e della request definite per il contratto del servizio. In sostanza, creare un web service, richiede la definizione di un file XML che rappresenta il contratto del servizio, la scrittura dei classici file di configurazione di Spring e la codifica delle classi di implementazione dell'endopoint e del servizio.

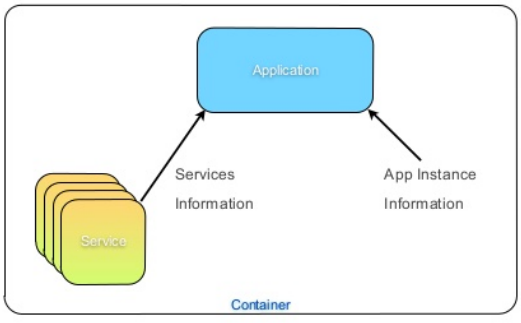
I vantaggi nell'uso di Spring-WS possono essere cosi ricapitolati:

- Velocizza l'utilizzo di Best Practise quali sviluppo mediante un approccio "contract-first", uso del profilo WS-I, basso accoppiamento tra contratto e implementazione;
- Potente mapping: è possibile distribuire ogni richiesta in entrata a qualsiasi componente in base al payload del messaggio, SOAP Header o espressioni XPath.
- Supporto per le API di XML: i messaggi XML in ingresso possono essere gestiti con JAXP API come DOM, SAX e STAX ma anche con JDOM, dom4j e XOM;
- Marshalling flessibile: è possibile usare JAXB 1 e 2, Castor, XMLBeans, JiBX e XStream;
- Spring-WS usa gli stessi principi alla base del framework Spring pertanto tutte le configurazioni possono essere definite nell'ambito dell'application context e questo riduce il tempo di sviluppo complessivo. L'architettura di Spring-WS è molto simile a quella di Spring-MVC;
- Supporto alla sicurezza: sono previste funzionalità per firmare i messaggi SOAP, cifrarli, decifrarli e autenticarli.

Per il rilascio del web service è sufficiente un lightweight container come Tomcat o Jetty.

SPRING CLOUD

Spring offre molti benefici anche in ambito Cloud. Una delle caratteristiche di Spring è la portabilità e negli scenari attuali tale qualità è sempre più importante. Per sfruttare i benefici del Cloud, all'interno dell'ecosistema Spring, è presente il modulo Spring Cloud che permette di creare rapidamente applicazioni per il Cloud che sfruttano alcuni pattern comuni nei sistemi distribuiti come gestione delle configurazioni, sessioni distribuite, stato del cluster, leadership election e service discovery. Tali applicazioni possono connettersi a vari servizi di cloud scoprendo tutte le informazioni necessarie a runtime.



Spring funziona in modo affidabile e coerente in tutte le PaaS (Platform as a service) dei più importanti vendor come Cloud Foundry, BeanStalk di Amazon Web Services, e Google App Engine. Una tipica applicazione fatta con Spring non richiederà modifiche per essere rilasciata,

ad esempio, su Cloud Foundry e pertanto potrà usare servizi come PostgreSQL, MySQL, MongoDB, Redis e RabbitMQ. La maggior parte delle offerte PaaS variano nelle tecnologie supportate ma un lightweight application server come Tomcat o Jetty è sempre presente. Sviluppando applicazioni con Spring è pertanto possibile sfruttare la modularità e l'agilità che le offerte PaaS forniscono. Due sono i concetti introdotti per permettere tale flessibilità: Cloud Connector e Service Connector. Un Cloud Connector è un'interfaccia che un fornitore di cloud implementa per permettere al resto della libreria di lavorare con la piattaforma di cloud. Un Service Connector è un oggetto che rappresenta la connessione a un servizio. E' possibile definire Cloud Connector e Service Connector personalizzati per supportare altre piattaforme e servizi cloud, rilasciarli come file Jar ed aggiungerli al classpath.

SPRING VS JEE

Come ampiamente detto, Spring Framework è stato introdotto con l'obiettivo di essere una semplice ed efficace alternativa agli EJB. Da quando Spring è stato proposto alla comunità degli sviluppatori sia il framework che gli stessi EJB si sono evoluti ed influenzati l'uno con l'altro. Nel Maggio 2013 gli EJB hanno raggiunto la versione 3.2 che associa alla semplicità di utilizzo un ricco set di funzionalità. Spring è alla versione 4.1.2 e da quando è uscito per la prima volta tantissime sono state le funzionalità che gli ideatori hanno aggiunto. Sebbene attualmente realizzare applicazioni con Spring o JEE comporti un tempo di sviluppo più limitato rispetto al passato e numerosi vantaggi, sul web esistono infinite diatribe tra sostenitori di Spring e di JEE. Per decidere quale stack tecnologico usare andrebbero presi in considerazione i seguenti punti:

- Requisiti non funzionali;
- Funzionalità offerte in accordo con le necessità del cliente;
- Competenze del team;
- Conseguenze che può provare la scelta di una determinata tecnologia rispetto ad un'altra.

Come si evince da discussioni e confronti presenti sul web i punti appena elencati, nella scelta tra un approccio basato su Spring o JEE, non vengono minimamente presi in considerazione. A tal proposito è utile effettuare un confronto tra Spring e JEE per analizzare eventuali punti di forza o criticità delle due tecnologie.

Comparazione

Feature	**EJB 3.x**	**Spring Framework**
Distributed Computing	• Permettono l'invocazione remota con RMI. Possibilità di accedere ai beans attraverso @Remote interfaces • Possibilità di effettuare un lookup remoto su JNDI • Accesso attraverso web services: JAX-WS, JAX-RS (non fanno parte degli EJB, ma EJBs possono essere acceduti usando tali servizi)	• Comunicazione remota con RMI, JAX-RPC, JAX-WS, JAX-RS • moduli: spring-ws, spring-webmvc
Beans	• @Stateless – fornisce logica di business	• Singleton – solo uno per IoC container; creato all'avvio

	• @Singleton – ne esiste solo uno per ogni applicazione • @Statefull – fornisce logica di business e può mantenere traccia sullo stato del client • @MessageDriven – permette di gestire messaggi	dell'applicazione • Prototype – creato ogni volta che si effettua una iniezione • Request – creato per una singola richiesta • Session –mantiene lo stato di una sessione http con un client • Global Session – mantiene lo stato in una sessione http globale
Gestione delle Dipendenze	• JNDI registry permette di ottenere beans usando interfacce di tipo local, remote o no-interface views • @EJB annotation permettono una iniezione delle dipendenze automatica • @Resource permette di iniettare risorse come EJBContext, TimerService, SecurityContext • possibilità di gestire dipendenze attraverso descrittori di deployment	• permette l'iniezione attraverso metodi setter, costruttori oppure mediante annotazioni del tipo @Autowired, @Resource, @Inject • inezione attraverso file di configurazione (chiamati application context descriptor) • permette il lookup in JNDI registry
Gestione della Concorrenza	• per i beans di sessione stateless e stateful session il container gestisce la concorrenza • il container gestisce un pool di beans accessibili • anche per bena singleton (dalla versione 3.1) è stato introdotto il supporto alla concorrenza • invocazione asincrona usando @Asynchronous	• tutti i beans sono singleton, beans con compiti di sessione sono creati per ogni client • la concorrenza è gestita dai bean • invocazione asincrona usando @Async
Gestione delle Transazioni	• il container gestisce le transazioni usando JTA • la gestione delle transazioni è permessa da container attraverso annotazioni o file di configurazione • nei bean @Statefull la gestione delle transazioni non deve essere esplicita	• gestisce le transazioni attraverso differenti API come JTA, JDBC, Hibernate, JPA, JDO • Supporto per una gestione dichiarativa delle transazioni
Messaging	• message driven come ascoltatori di messaggi • supporto per code (point2point) e topic (publish-subscribe)	• con spring-jms è possibile creare dei consumatori e produttori di messaggi • richieste un MQ esterno come

	• concorrenza gestita dal container	ActiveMQ
Aspect Oriented Programming	• Interceptors 1.1 (since 3.1)	• Modulo spring-aop e AspectJ, CGLIB
Scheduling	• scheduling jobs usando TimerService • @Scheduled	• Spring TaskScheduler • Quartz può essere aggiunto a Spring
Security Management	• Integrated support for declarative and programmatic security through JAAS	• modulo spring-security ricco di funzionalità relative alla sicurezza
Integration Testing	• integration testing con Arquilian • integration testing con embedded container	• modulo dedicato al testing ricco di funzionalità
Deployment	• enterprise archive *.ear –può consistere di molti moduli web ed ejb • singolo modulo web *.war • configurazione in application.xml (*.ear) e ejb-jar.xml (*.jar, *.war) • richiede appplication server per EJB come JBoss, WebLogic, Apache Tomee/+	• packaged in *.jar or *.war files • configurazione in *.xml files oppure mediante annotazioni • rilascio in lightweight containers come Tomcat, Jetty

Cosa usare?

Capire quando è meglio usare Spring o JEE è cosa assai ardua specialmente oggigiorno. Entrambe le soluzioni forniscono mezzi per raggiungere gli stessi obiettivi pertanto per decretare quale tecnologia usare nella realizzazione di un sistema i veri punti da prendere in considerazione sono i seguenti:

- enviroment (standalone o distribuito);
- esigenze funzionali del cliente;
- qualità non funzionali (scalabilità, fail-over);
- conoscenza generale del modello di programmazione per entrambe le soluzioni;
- competenze del team di sviluppo.

Adottare un approccio basato sugli EJB attualmente non è più complesso come poteva essere agli albori di tale tecnologia. Realizzare un'applicazione che richiede diversi moduli di Spring può portare ad avere un risultato finale che è formato da diverse centinaia di megabyte di codice.