**MAVEN**

**Introduzione**



Maven è un potente strumento open source per la gestione dei progetti Java, in termini di compilazione del codice, distribuzione, documentazione e collaborazione del team di sviluppo. Maven è basato sul concetto di Project Object Model (POM), un file XML, che descrive le dipendenze tra il progetto e le varie versioni di librerie nonché standardizza la struttura delle directory. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie. Apprendere il suo uso può semplificare di molto il difficile compito di compilazione e di gestione dei progetti Java viste le numerose funzionalità che esso offre. Ne deriva un'estrema semplificazione del processo: con Maven non è più necessario conoscere nel dettaglio i meccanismi di complilazione.

I vantaggi principali di Maven sono i seguenti:

* Standardizzazione della struttura di un progetto e del processo di compilazione.
* Test ed esportazione automatizzati.
* Gestione e download automatico delle librerie necessarie al progetto con risoluzione delle eventuali dipendenze transitive.
* Facilità di ampliarne le funzionalità iniziali tramite l’utilizzo di plugin.
* Creazione automatica di un semplice sito di documentazione del progetto.
* Molto altro ancora.

**Breve storia di Maven**

L'idea iniziale, come spesso accade nel mondo dell'Open Source è nata dall'esigenza di risolvere un problema pratico. La realizzazione di Maven è partita nell'ambito del progetto Jakarta Alexandria (attualmente abbandonato) per poi migrare all’interno del progetto Turbine (framework Servlet disegnato per la rapida produzione di applicazioni web). L'obiettivo iniziale era di implementare un tool per semplificare, uniformare e automatizzare il processo di build di sistemi complessi. Per ottemperare a tali scopi serviva sia creare un modello di progetto, sia una struttura di file system standard. Era necessario anche fare in modo che i diversi progetti Apache funzionassero in maniera analoga. Prima del rilascio di Maven, infatti, ciascun progetto Apache, presentava differenti approcci alla compilazione, alla distribuzione e alla generazione del sito web relativo al progetto, creando evidenti problemi di allocazione delle risorse umane, di riutilizzo di script, di implementazione di best practice. Questi problemi non erano di poco conto, visto il molto tempo speso dagli sviluppatori per configurare i vari ambienti, per comprenderne il funzionamento e per mantenere i vari script di compilazione. Oltre a creare elementi di distrazione dal loro obiettivo principale (realizzare sistemi software di elevata qualità), tutto ciò spesso finiva anche per far desistere nuovi potenziali collaboratori. Dato l'elevato livello di interdipendenza dei progetti open-source, la capacità di Maven di standardizzare l'ubicazione dei vari file (sorgenti, documentazione, distribuzione, etc.), di fornire una struttura comune per i vari progetti, di permettere il reperimento dei file di distribuzione attraverso un meccanismo di repository condiviso e di far sì che il processo di build diventasse più agevole, trasparente e meno complesso, ha rappresentato un elemento chiave che ne ha assicurato il successo fin dalle prime versioni. Già Ant (strumento simile a Maven) aveva risolto molti problemi relativi al processo di build; Maven permise di effettuare l'ulteriore passo in avanti, risolvendo le rimanenti problematiche presenti nello sviluppo di sistemi complessi basati su sottoprogetti interdipendenti:

* eliminazione di diversi script di build da eseguire secondo un rigido ordine;
* eliminazione della necessità di copiare porzioni di file di script da un progetto a un altro (con la conseguente necessità di doverne gestire una moltitudine);
* rimozione della necessità di copiare i vari file jar, soprattutto di quelli prodotti dai vari sotto-progetti;
* risoluzione dei problemi relativi alla condivisione delle librerie (diverse versioni, mancanza di allineamento): Maven rende possibile disporre di un progetto "genitore" dal quale far ereditare sotto-progetti.

Dalla versione iniziale, Maven ha fatto molti progressi e ha contribuito enormemente a semplificare le quotidiane attività del team di sviluppo.

**Obiettivi primari di Maven**

Gli obiettivi principali di Maven sono:

1. Semplificazione del processo di build dei sistemi Java. In particolare, Maven si fa carico di risolvere tutta una serie di dettagli senza ricorrere all'utilizzo di file di script,
2. Sviluppo di un ambiente uniforme di build. Maven gestisce progetti basati sul proprio modello a oggetti del progetto (POM, Project Object Model). Sebbene molti aspetti di Maven possano essere personalizzati, Maven dispone di una serie di "standard", studiati per essere efficacemente utilizzati per i vari progetti. Pertanto, una volta compreso il funzionamento di un progetto gestito da Maven, si sa come gestirne quasi ogni altro (a meno di forti personalizzazioni). Il che si traduce in un notevole risparmio di tempo, energie e frustrazione.
3. Produzione di informazioni qualitative circa il progetto. Sebbene Maven non sia né uno strumento di documentazione né un generatore di siti web, rappresenta un'ottima infrastruttura che permette di utilizzare tutta una serie di plug-in per la generazione di informazioni utili relative ai progetti. Per esempio, permette di generare documenti relativi alle variazioni effettuate (interagendo con i sistemi di source control), ai riferimenti incrociati dei sorgenti, alle mailing list, alle dipendenze, ai rapporti relativi alla copertura del codice da parte dei test, etc. Il tutto in maniera assolutamente trasparente e automatica.
4. Erogazione di linee guida corredate da un opportuno supporto per l'applicazione di best practice per lo sviluppo di sistemi. L'esempio più evidente è relativo alla presenza esplicita, nel processo di build standard, della definizione, specifica ed esecuzione dei test di unità. Inoltre vi sono workflow necessari per il rilascio e la distribuzione di nuove release del sistema.
5. Supporto alla migrazione, quanto mai trasparente, verso nuove feature.

**Caratteristiche**

Le principali caratteristiche offerte da Maven sono

* setup semplificato dei progetti in rispetto alle best practices "implementate": preparare un nuovo progetto o estrapolare un modulo richiede un tempo inferiore al minuto;
* utilizzo consistente indipendente dai singoli progetti, il che equivale a minimizzare la curva di apprendimento;
* gestione avanzata delle dipendenze corredata da aggiornamento automatico e gestione delle dipendenze transitive (A -> B, B -> C => A -> C);
* gestione simultanea di diversi progetti;
* continua espansione della già considerevole disponibilità di plug-in, librerie e meta-dati da utilizzarsi per la gestione dei propri progetti: molti di questi elementi, inoltre, sono open-source e la relativa comunità è molto attiva;
* estensibilità, è possibile non solo modificare le impostazioni di default, ma anche scrivere propri plug-in;
* immediato accesso alle nuove feature con minimo dispendio di tempo;
* disponibilità di task Ant per la gestione delle dipendenze e per il deployment esterno all'ambiente Maven;
* build basati su modello: Maven è in grado di eseguire il build di una serie di progetti e di includere le relative distribuzioni in appositi file .jar, .war etc. senza dover ricorrere ad nessuno script;
* sito coerente di informazioni relative al progetto: a tal fine Maven utilizza gli stessi dati utilizzati dal processi di build; Maven, inoltre è in grado di generare rapporti in una serie di formati, incluso il PDF;
* gestione del processo di rilascio di nuove versioni e pubblicazione dei file di distribuzione: ciò avviene in maniera quasi trasparente anche in presenza di sistemi di Control Managment come CVS e ClearCase;
* gestione delle dipendenze: Maven incoraggia l'utilizzo di repository sia locali sia remoti, per la memorizzazione dei file di distribuzione. Inoltre, dispone di una serie di meccanismi che permettono di eseguire il download, da un sito globale, di specifiche librerie richieste dal proprio progetto: questo semplifica il riutilizzo degli stessi file .jar da parte di diversi progetti, fornendo anche informazioni necessarie per gestire problemi relativi alla retrocompatibilità (backward compatibility).

**Vantaggi derivati dall'utilizzo di Maven in un progetto**

Date le caratteristiche di Maven evidenziate nei paragrafi precedenti, dovrebbero essere ormai chiari quali siano i vantaggi derivanti dall'utilizzo nella gestione dei progetti software, specie in quelli non semplici. In particolare:

* **Coerenza**: le varie organizzazioni possono standardizzare la gestione dei progetti Java utilizzando l'insieme di best practice alla base di Maven. Accettando una serie di comodi standard, si accede a tutta una serie di servizi predefiniti. Il livello di qualità dei vari progetti, inevitabilmente, si eleva; i progetti stessi diventano quindi più trasparenti, si minimizza il tempo necessario per comprendere i vari progetti e quindi si facilita il movimento delle risorse umane.
* **Riutilizzo**: si tratta uno degli elementi alla base di Maven, il cui uso, di fatto è già esso stesso un primo riutilizzo di best practice. Un ulteriore livello di riutilizzo è garantito dal fatto che la business logic è incapsulata in comodi moduli (plug-in);
* **Maggiore agilità**: utilizzando Maven si semplifica il processo di generazione di nuovi componenti, di integrazione tra componenti, di condivisione di file eseguibili, inoltre, la curva di apprendimento di ciascun progetto viene incredibilmente ridotta.
* **Semplificazione della manutenzione**: non è più necessario investire tempo e risorse per manutenere gli ambienti e script di build, i quali, oltre ad essere standardizzati, sono gestiti da Maven.

**I principi base di Maven**

L'idea base durante la progettazione di Maven era di creare un linguaggio condiviso per lo sviluppo di progetti software basati su Java ovvero ciò che Christopher Alexander ha formalmente definito pattern. Avere un linguaggio condiviso ad alto livello permette agli sviluppatori di trattare ad elevato livello di astrazione quei progetti la cui struttura sia fortemente standardizzata. Ne consegue una rete di comunicazioni più efficaci, una serie di servizi automatizzati e quindi un miglioramento della qualità generale e della produttività del progetto. Definire un pattern per lo sviluppo dei progetti software richiedeva i seguenti principi base:

1. convenzioni sulla configurazione (convention over configuration) e in particolare:

1.1. organizzazione standard della directory dei progetti (ubicazione delle risorse del progetto, dei file di configurazione, di quelli generati, della documentazione, etc.);

1.2. definizione del vincolo base secondo cui un progetto Maven genera un solo output (file di distribuzione). Ciò porta naturalmente alla realizzazione di file pom.xml con un obiettivo limitato e ben definito, e quindi all'applicazione del principio di separazione delle responsabilità;

1.3. convenzione sui nomi;

2. esecuzione dichiarativa;

3. riutilizzo di logica di build;

4. organizzazione coerente delle dipendenze.

Per quanto concerne il primo punto, l'idea base, molto semplice, è che se si accettano una serie di valide convenzioni (frutto di lungo studio ed esperienza), si ottiene una serie di vantaggi, non solo legati al fatto che lo stesso problema è già stato ben analizzato e validamente risolto. Per esempio, se si organizza la struttura dei propri progetti in base allo standard Maven, quest'ultimo è in grado di fornire automaticamente una serie di servizi, come la generazione dei file di progetto per i vari IDE, la compilazione automatica, etc.   
Convention over configuration non è altro che la ripetizione di una serie di semplici regole di buon senso relative all'applicazione di standard. In particolare, l'applicazione di standard fa risparmiare tempo, semplifica la comunicazione, facilita il reperimento di risorse, permette di creare ulteriore valore aggiunto partendo da solide infrastrutture, permette di condividere il lavoro, e così via.  
Per quanto riguarda l'esecuzione dichiarativa (punto 2) basti dire che tutto in Maven ha una natura dichiarativa (che tuttavia, volendo, può essere alterata). L'elemento base di Maven, il file pom.xml, è probabilmente l'esempio più classico di struttura dichiarativa. Con una decina di righe di carattere dichiarativo (e non procedurale), è possibile compilare, verificare, generare la documentazione e il file di distribuzione di semplici progetti.

Per quanto attiene il punto 3, Maven semplifica enormemente il riutilizzo del codice, incapsulando la business logic in appositi moduli: i plug-in, di cui esiste un insieme base (come quello per compilare, per eseguire i test, per creare file .jar, etc.) oltre a un insieme vastissimo di plug-in forniti da terze parti.   
Per quanto riguarda poi l'organizzazione coerente delle dipendenze (punto 4), questa è ottenuta attraverso una serie di meccanismi di repository. In primo luogo esiste una sezione all'interno del file pom.xml, dedicata alla dichiarazione delle dipendenze. Una volta definita una nuova dipendenza nel pom.xml, non è necessario copiare il relativo file in un'apposita directory del progetto, poiché risiederà in un repository. La presenza dei repository risolve una serie di problemi tipici concernenti l'indicazione dei path, le diverse versioni delle librerie e così via.   
Maven dispone di due tipi di repository: locale e remoto. Durante il normale funzionamento, Maven, interagisce con il repository locale; qualora però una dipendenza non sia presente all'interno di tale repository, Maven si occupa di consultare i repository remoti ai quali ha accesso, al fine di risolvere la dipendenza mancante. Qualora non si definisca diversamente, l'ultimo repository acceduto in ordine di tempo è il repository globale presso ibiblio (http://mirrors.ibiblio.org/pub/mirrors/maven2/). Questa consultazione serve per individuare, e quindi scaricare nei vari repository (dapprima quelli remoti e poi automaticamente in quelli locali) i file necessari per risolvere la dipendenza dichiarata (tipicamente degli archivi JAR).

**Concetti di Maven**

Di seguito alcuni concetti basilari di Maven utilizzati nel lavoro di tesi.

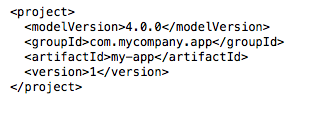
**File di POM**

**POM** sta per Project Object Model e ogni singolo progetto è descritto attraverso il file di configurazione pom.xml, senza il quale Maven non può fare niente. Il POM è un file XML e guida l’esecuzione in Maven definendo in modo chiaro l’identità e la struttura di un progetto in ogni suo aspetto. Tutto è descritto nel POM: informazioni generali  del progetto, dipendenze, processo di compilazione e fasi secondarie come la generazione di documentazione.

Un POM ai minimi termini è composto da un tag root <project> che conterrà i tag:

* <modelVersion> che dichiara a quale versione di POM questo progetto è conforme (a noi ci basta sapere che deve essere settato a 4.0.0)
* <groupId> ID del gruppo del progetto
* <artifactId> ID dell’artefatto (del progetto)
* <version> cioè la versione del progetto
* <packaging> che è il tipo di archivio che vogliamo esportare (**jar**, **war** o **ear**, come vedremo troveremo in una cartella chiamata “**target**” l’archivio esportato)

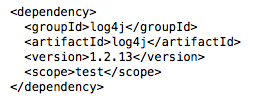
I parametri **groupId**, **artifactId**, **version** e **packaging** identificheranno univocamente un progetto. Se packaging non è specificato nel POM, assumerà “jar” a valore di default.  
Ecco un esempio di pom.xml minimale con packaging non specificato:



Per ogni dipendenza è possibile anche definire uno scope:

* **compile** (default) – le dipendenze sono disponibili in tutti i classpath del progetto
* **provided** – è simile a compile, ma prevede che a runtime le dipendenze siano rese disponibili dall’ambiente di esecuzione (per esempio le JavaEE APIs per un’applicazione enterprise)
* **runtime** – le dipendenze sono richieste solo in esecuzione
* **test** – le dipendenze sono richieste solo per la compilazione e l’esecuzione dei test
* **system** – la dipendenza non viene recuperata tramite repository, ma ne viene esplicitamente dichiarata la posizione locale

Ad esempio:



**Plugin e Goal**

Un **goal** è una singola funzione che può essere eseguita sul progetto, l’equivalente Maven dei task Ant. I Goal possono essere sia specifici per il progetto dove sono inclusi, sia riusabili. I **Plugin** sono goal riutilizzabili in tutti i progetti.  
Maven ha una serie di plugin built-in disponibili, i principali sono i seguenti:

* **clean**: che permette di cancellare i compilati dal progetto;
* **compiler**: che permette di compilare i file sorgenti;
* **deploy**: che permette di depositare il pacchetto generato nel repository remoto;
* **install**: che permette di depositare il pacchetto generato nel repository locale;
* **site**: che permette di generare la documentazione del progetto;
* **archetype**: che permette di generare la struttura di un progetto a partire da un template.

Ciascun plugin mette a disposizione dei goal specifici o più di un goal (per esempio, il plugin di compilazione, **compiler**, prevede un goal per la compilazione del codice sorgente e un altro per la compilazione dei casi di test). Ciascun goal, a sua volta, riceve in ingresso specifici parametri, facoltativi o obbligatori.  
Altri plugin possono essere realizzati qualora sia necessario estendere ulteriormente le capacità di Maven a causa di particolari esigenze.

**L’archetipo di Maven**

Una delle caratteristiche particolarmente apprezzate di Maven è quella di fornire un insieme di standard che rendono possibile l’applicazione di tutta una serie di "buone pratiche". Uno di questi standard è costituito dalla struttura della directory del progetto, denominata archetype (archetipo). Coerentemente con l’impostazione di Maven, applicare tale standard non è obbligatorio, sebbene sia consigliabile: Maven fornisce una serie di strumenti che permettono di utilizzare strutture diverse e, in casi molto particolari, è anche possibile non conformarsi agli standard. Ma occorre comunque ricordare che deviare dalle impostazioni standard tende a invalidare, o comunque a ridurre, la portata di tutta una serie di vantaggi, quali quelli tipici derivanti dal fatto che si tratti di uno “standard”:

i vari servizi di Maven sono predisposti per funzionare con questa struttura e quindi nessuna ulteriore operazione è richiesta per farli funzionare;

1. tale standard permette di uniformare i vari progetti presenti all’interno di un’organizzazione;
2. tale struttura semplifica l’apprendimento dei progetti e quindi favorisce lo spostamento del personale;
3. si evitano inutili perdite di tempo legate a discussioni relative alla struttura del progetto ("dove posizioniamo il file X ?"), alla necessità di creare nuove directory per memorizzare determinati file, e così via.

Non va dimenticato poi che ci sono vantaggi legati al fatto che l’archetipo è il risultato di un attento e prolungato studio. Quindi, invece di cercare di inventarsi l’ennesima struttura, con il rischio peraltro di ottenere risultati modesti o addirittura problematici, è decisamente più facile e conveniente beneficiare di un disegno derivante da una lunga esperienza che include anche l’uso di plug-in e di svariati tool. Pertanto, l'utilizzo della struttura standard elimina alla radice il rischio di utilizzare un'organizzazione delle cose poco adatta all'uso di importanti tool di sviluppo/plug-in.

**Ereditarietà**

Un’altra caratteristica molto potente e unica di Maven è la possibilità di relazionare tra loro i progetti con **legami di ereditarietà**, un po’ come avviene con le classi Java.  
In Maven l’ereditarietà permette di creare nuovi **POM** file che ereditano le relazioni definite in un altro POM genitore detto **super POM**. I progetti genitori e, come abbiamo visto, quelli esclusivamente aggregabili devono specificare il valore pom all’interno del campo packaging. Questo campo serve in generale per indicare la natura del prodotto finale: **jar**, **war**, **ear**, ma anche **Maven-plugin**, **rar**, etc… A meno che il valore non corrisponda a pom, nel qual caso si indicano progetti “genitore” o aggregabili.  
Gli elementi ereditabili dai discendenti che si possono specificare in un POM genitore sono:

* dipendenze
* lista degli sviluppatori e contributori
* esecuzioni dei plugin con id corrispondenti
* configurazione dei plugin
* risorse

**Maven ed Eclipse**

E’ possibile utilizzare Maven in Eclipse senza ricorrere all’istallazione di Maven sul computer. Esiste infatti un plugin chiamato m2e che permette di sfruttare la potenza di Maven in Eclipse. E’ possibile trovare ed installare il plugin m2e sul marketplace.

<http://www.cosenonjaviste.it/organizziamoci-con-maven-parte-i/>

http://www2.mokabyte.it/cms/article.run?articleId=S85-L5J-HP3-86O\_7f000001\_30480431\_0844866c