

INTRODUZIONE

Framework?

Un framework, è un'architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, col fine di facilitare lo sviluppo al programmatore.

Perchè usare un framework?

Standard -> Metodologia

Community -> Supporto

Strumenti -> IDE e Plugin

2.

SPRING &

SPRING BOOT

SPRING - INTRODUZIONE

In informatica Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java. (“Wikipedia”)

Spring è un framework molto datato (2002) ma ancora in sviluppo e largamente diffuso.

Sposta il focus della programmazione sugli aspetti, sul comportamento e non su come deve fare, a quello ci pensa il framework (Inversion of Control - IoC).

Implementa design pattern, come la Dependency Injection, per la condivisione e il riuso degli oggetti.

INVERSION OF CONTROL

L'Inversion of Control è un principio architetturale nato alla fine degli anni ottanta, basato sul concetto di invertire il controllo del flusso di sistema (Control Flow) rispetto alla programmazione tradizionale.

Programmazione tradizionale: lo sviluppatore definisce la logica del flusso di controllo, specificando le operazioni di creazione, inizializzazione degli oggetti ed invocazione dei metodi.

IoC: Si inverte il control flow, facendo in modo che non sia più lo sviluppatore a doversi preoccupare di questi aspetti, ma il framework, che reagendo a qualche "stimolo" se ne occuperà per suo conto.

Questo principio è anche conosciuto come Hollywood Principle: "Non chiamarci, ti chiameremo noi".

DEPENDENCY INJECTION

Utilizzato per riferirsi ad una specifica implementazione dello IoC

Dipendenze iniettate a run-time dall'esterno della classe, tramite:

- Uso del costruttore
- Uso di una Factory
- Servizio di naming

La classe A dipende da B se ne usa un'istanza o dei metodi.

Soluzione

Un componente esterno (assembler) si occupa della creazione degli oggetti e delle loro relative dipendenze.

Gli oggetti sono assemblati e le dipendenze risolte con l'utilizzo dell'injection tramite Constructor Injection (passata dal costruttore), Setter Injection (settata con metodo set) o Interface Injection (mapping tra interfaccia e implementazione, non usata in Spring).

SPRING - LAYERS

Spring framework si compone da moduli (layers) le cui attività sono ben definite (Core, Beans, Context, ecc...).

Core Container Core Beans Context Expression Language	Data Access JDBC ORM OXM JMS Transaction	I moduli Core e Beans formano la struttura principale del framework, includendo le funzionalità di IoC e DI. L'elemento fondamentale è la BeanFactory.
Web Web Web-Servlet Web-Struts Web-Portlet	AOP Aop AspectJ	Il layers Data Access fornisce un'astrazione per le API JDBC, un modulo per integrare JPA e diversi ORM (Hibernate).
Test		Il layers Web offre un server tomcat e tecnologie di view e MVC.
		Il layers di AOP implementa lo standard AOP Alliance per l'aspect oriented programming.

SPRING - Beans

L'IoC container è la parte di Spring che si occupa di instanziare e configurare gli oggetti che vengono inseriti in esso, creando così l'ApplicationContext, che prendono il nome di **beans** (oggetti, come ad esempio @Component, creati da Spring).

I beans vengono configurati attraverso dei metadati che possono essere dei file xml o delle Java annotations o Java Configuration.

@Component



@Controller



@Service



@Repository

SPRING - MVC Framework

La parte di Spring che si occupa del layer di presentazione dei dati specifico per applicazioni Web è il Web MVC Framework.

Come da filosofia di Spring anche per usare questo modulo non è necessario estendere o implementare nessuna interfaccia specifica del framework.

Per implementare l'MVC, Spring usa tre elementi principali:

- **DispatcherServlet**: smista le richieste in ingresso POST, GET, PUT, ... e instrada le risposte
- **Handlers**: usa i controller (POJO annotati da @Controller) popola un model con i risultati
- **View Resolver**: componente di view (jsp o json) prepara la risposta con i dati ottenuti dal model per essere rappresentata

SPRING BOOT

La generazione di un progetto web segue spesso schemi ben precisi e ridondanti. Per rendere meno tedioso e ridurre gli errori i developer hanno a disposizione un tool per creare un nuovo progetto Spring con tutto ciò che serve preconfigurato.

Spring Boot è un'applicazione Spring con numerose configurazioni impostate per semplificare l'avvio di un progetto.

La gestione di progetto spring è demandata a strumenti secondari per curara la compilazione del codice, la gestione delle dipendenze, la generazione della documentazione e organizzare la collaborazione in team.

I più diffusi strumenti a supporto di Spring sono indubbiamente Maven e Gradle.

SPRING BOOT

La generazione di un progetto web segue spesso schemi ben precisi e ridondanti. Per rendere meno tedioso e ridurre gli errori i developer hanno a disposizione un tool per creare un nuovo progetto Spring con tutto ciò che serve preconfigurato.

Spring Boot è un'applicazione Spring con numerose configurazioni impostate per semplificare l'avvio di un progetto.

La gestione di progetto spring è demandata a strumenti secondari per curara la compilazione del codice, la gestione delle dipendenze, la generazione della documentazione e organizzare la collaborazione in team.

I più diffusi strumenti a supporto di Spring sono indubbiamente Maven e Gradle.

SPRING BOOT

Tool per la generazione di un progetto SpringBoot:

<http://start.spring.io>

Spring Boot vs Spring

<https://dzone.com/articles/spring-boot-vs-spring-mvc-vs-spring-how-do-they-compare>



3.

MAVEN

INTRODUZIONE

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

<https://maven.apache.org/>

OBIETTIVI DI MAVEN

L'obiettivo principale di Maven è di aiutare lo sviluppatore nel gestire e comprendere a pieno lo stato di un progetto nel minor tempo possibile.

Inoltre Maven permette di:

- facilitare i processi di build
- fornire un sistema di build uniforme
- fornire informazioni di qualità sul progetto
- fornire linee guida sulle best practices in fase di sviluppo

<https://maven.apache.org/what-is-maven.html>

COMPONENTI DI MAVEN

- **pom.xml**: file xml di configurazione, contiene tutte le informazioni sul progetto (dipendenze, test, documentazione, etc...).
- **Goal**: singola funzione che può essere eseguita sul progetto, l'equivalente Maven dei task in Ant.
- **Plug-in**: goal riutilizzabili in tutti i progetti.
- **Repository**: directory strutturata destinata alla gestione delle librerie / dipendenze. Può essere locale o remoto.

POM.XML

POM (Project Object Model) descrive ogni progetto attraverso il file di configurazione pom.xml. 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.*

POM.XML

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

- <modelVersion> dichiara a quale versione di POM il progetto è conforme (attuale 4.0.0)
- <groupId> ID del gruppo del progetto (solitamente organizzazione)
- <artifactId> ID dell'artefatto (nome del progetto)
- <version> cioè la versione del progetto
- <packaging> indica il tipo di archivio che vogliamo esportare (jar, war o ear)

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

POM.XML

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3
4           .org/2001/XMLSchema-instance"
5               xsi:schemaLocation="http://maven.apache
6               .org/POM/4.0.0 http://maven.apache
7               .org/xsd/maven-4.0.0.xsd">
8       <modelVersion>4.0.0</modelVersion>
9       <groupId>it.nextre.academy</groupId>
10      <artifactId>spring-jpa-hibernate-demo1
11      </artifactId>
12      <version>0.0.1-SNAPSHOT</version>
13      <packaging>jar</packaging>
14  </project>
```

POM.XML

La grande forza di Maven è la gestione automatizzata delle dipendenze, e delle relative versioni, lette dal repository. Le dipendenze vanno specificate all'interno del nodo `<dependecies>` del file pom. Ogni dipendenza è composta da `groupId`, `artifactId` e `version` dentro al nodo `<dependency>`.

```
<project>
  ...
<dependecies>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.13</version>
  </dependency>
</dependecies>
  ...
</project>
```

POM.XML

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

POM.XML

Il **Repository** è una directory strutturata destinata alla gestione delle librerie. Maven ne crea uno in locale sotto la home dell'utente al fine di evitare accessi alla rete inutili.

Nel processo di gestione del progetto Maven verifica in locale l'esistenza della libreria desiderata e in caso negativo interroga un repository remoto.

Nel POM minimale non è specificato alcun repository remoto.

Se si svolge il building del progetto, allora verrà ereditato quello di default che corrisponde a:

<http://repo1.maven.org/maven2/>

POM.XML

Qualora fosse necessario, è però possibile specificare altri *repository* dove cercare librerie e dipendenze. Per integrarlo all'interno del nostro progetto è sufficiente aggiungere nel pom.xml la sezione <repositories>

```
<project>
...
<repositories>
    <repository>
        <id>id_nuovo_repository</id>
        <url>http://url_nuovo_repository</url>
    </repository>
</repositories>
...
</project>
```

POM.XML

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.

POM.XML

Un'altra caratteristica molto potente e unica di Maven è la possibilità di relazionare tra loro i progetti con legami di **ereditarietà**. In Maven l'ereditarietà permette di creare nuovi POM file che ereditano le relazioni definite in un altro POM genitore detto *super POM*.

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

POM.XML

Qualora si intenda ereditare da un altro file pom.xml le relative coordinate di questo genitore (groupId, artifactId, version), esse vanno specificate all'interno della sezione <parent>, che dispone di un ulteriore campo non obbligatorio: <relativePath>. Questo indica il path in cui cercare il progetto genitore, prima di accedere al repository locale e infine a quello remoto.

```
<project>
...
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.4.RELEASE</version>
    <relativePath/>
</parent>
...
</project>
```

POM.XML

Altre informazioni utili e presenti in un pom.xml

- **name:** permette di specificare il nome “interno” del progetto, utilizzato dal team di sviluppo;
- **description:** una breve descrizione del progetto;
- **url:** indirizzo del sito dedicato al progetto;
- **inceptionYear:** indica l’anno di startup del progetto.
- **licenses:** permette di indicare una o più licenze con la quale verrà rilasciato il progetto. Per ogni singola licenza, è possibile specificare: name, url, comments e distribution[repo|manual].
- **organization** Informazioni relative all’azienda/organizzazione proprietaria del progetto.
- **properties:** definisce l’elenco delle singole property che possono essere dinamicizzate nel progetto.
- **build:** dichiarazione delle specifiche di build, per profilo e/o per progetto
- **build -> resource:** indica i file da aggiungere al progetto senza essere compilati
- **build -> plugins:** definizione dei plugin

```
<properties>
    <java.version>1.8
</properties>
```

POM.XML

Altre informazioni utili e presenti in un pom.xml

- **name**: permette di specificare il nome “interno” del progetto, utilizzato dal team di sviluppo;
- **description**: una breve descrizione del progetto;
- **url**: indirizzo del sito dedicato al progetto;
- **inceptionYear**: indica l’anno di startup del progetto.
- **licenses**: permette di indicare una o più licenze con la quale verrà rilasciato il progetto. Per ogni singola licenza, è possibile specificare: name, url, comments e distribution[repo|manual].
- **organization** Informazioni relative all’azienda/organizzazione proprietaria del progetto.
- **properties**: definisce l’elenco delle singole property che possono essere dinamicizzate nel progetto.
- **build**: dichiarazione delle specifiche di build, per profilo e/o per progetto
- **build -> resource**: indica i file da aggiungere al progetto senza essere compilati
- **build -> plugins**: definizione dei plugin

```
<properties>
    <java.version>1.8
</properties>
```

SITOGRAFIA MAVEN

Un elenco di risorse per approfondire quanto visto in precedenza

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

<http://www.html.it/articoli/maven-organizzazione-dei-progetti-java-1/>

<https://www.sonatype.com/ebooks>

<http://www.valeriofinazzo.it/joomla/maven/>

<http://codingjam.it/maven-sveliamo-un-pom-di-segreti/>



4.

REST API

API

Con *application programming interface* (in acronimo API, in italiano *interfaccia di programmazione di un'applicazione*), in informatica, si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma.

[Wikipedia]

- Astrazione di alto livello
- Riuso del codice
- Modularità
- Stratificazione delle operazioni
- Interagire con sistemi informatici

API

API danno la possibilità di usare sistemi e contenuti proprietari al di fuori dell'ambiente principale per il quale sono state progettate.

In alcuni contesti le stesse API vengono usate solo internamente per estendere con semplicità e coerenza le funzionalità dell'applicativo iniziale.

Garantiscono la separazione dei contenuti dalla loro rappresentazione.

API private (restricted)

API pubbliche (public)

Problematiche

- Gestione delle informazioni
- Perdita di controllo diretto
- Aterritorialità
- Cambio di finalità

REST

REST = architettura per la progettazione di un sistema, non è legato a un protocollo in particolare ma a un modo di progettare concreto, ben definito e standardizzato, di cui non esistono linee guida implementative, ma solo un insieme di linee guida per la realizzazione di una “architettura di sistema”. Con la filosofia REST si vogliono esporre delle risorse e la loro elaborazione.

L'architettura REST non è legata solamente al web, anzi solo negli ultimi anni, con l'avvento dei web service, è diventata famosa.

Principi da rispettare

- Identificazione delle risorse
- Utilizzo esplicito dei metodi
- Risorse autodescrittive
- Collegamenti tra risorse
- Comunicazione senza stato

REST

Identificazione delle risorse

Si intende un qualsiasi elemento oggetto di elaborazione. Per fare qualche esempio concreto, una risorsa può essere un cliente, un libro, un articolo, un qualsiasi oggetto su cui è possibile effettuare operazioni.

Ciascuna risorsa deve essere identificata univocamente (URI).

Risorse autodescrittive

Il formato dell'URI deve descrivere la risorsa e il singolo elemento

Utilizzo esplicito dei metodi

Utilizzare al meglio il protocollo per rappresentare le operazioni sulle risorse (non per forza HTTP)!

Le risorse REST sono stateless

REST API

Identificato il protocollo HTTP per la comunicazione si sfruttano i suoi metodi per mappare delle azioni da compiere sulle risorse. Tipicamente i metodi scelti sono **GET**, **POST**, **PUT** e **DELETE**.

In un contesto RESTful sappiamo già a priori come ottenere la rappresentazione di una risorsa. In altre parole, questo principio REST stabilisce una mappatura uno a uno tra le tipiche operazioni CRUD (creazione, lettura, aggiornamento, eliminazione di una risorsa) e i metodi HTTP.

Metodo HTTP	Operazione CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene una risorsa esistente
PUT	Update	Aggiorna una risorsa o ne modifica lo stato
DELETE	Delete	Elimina una risorsa

REST API

Le risorse restituite non sono direttamente i record di una tabella ma la loro rappresentazione dipende dall'implementazione del web service, anche il formato può dipendere dalla richiesta del client e/o dall'implementazione del servizio.

I principi REST non impongono vincoli su come rappresentare le risposte, ma è bene affidarsi ad uno standard (JSON o XML). Il tipo di risposta può essere richiesto dal client durante la richiesta della risorsa.

```
GET /clienti/1234
HTTP/1.1
Host: www.myapp.com
Accept: application/vnd.myapp.cliente+xml
```

Esempio di richiesta proveniente da GET per ottenere la risorsa clienti con id 1234 la cui risposta accettata dal client deve seguire il formato XML con DTD vnd.myapp.cliente.

REST API

Le risorse restituite devono essere collegate tra di loro, questo perché la struttura della API si basa sul concetto di autoconsistenza, pertanto è bene sfruttare gli URI per accedere alle risorse collegate, semplicemente seguendo i link delle risorse restituite.

```
<ordine>
  <numero>12345678</numero>
  <data>01/07/2011</data>
  <cliente rif="http://www.myapp.com/clienti/1234" />
  <articoli>
    <articolo rif="http://www.myapp.com/prodotti/98765" />
    <articolo rif="http://www.myapp.com/prodotti/43210" />
  </articoli>
</ordine>
```

Ogni richiesta è **stateless**, dato che il protocollo HTTP non ha una storia delle richieste, ciascuna richiesta non ha alcuna relazione con le richieste precedenti e successive. Lo stesso principio si applica ad un Web Service **RESTful**, cioè le interazioni tra client e server devono essere senza stato. È vietato tenere traccia dello stato lato server, l'applicazione può fare ciò che vuole.

REST API - SICUREZZA

La sicurezza di un sistema REST coinvolge i seguenti aspetti:

- **l'autenticazione**, cioè la capacità di identificare le parti coinvolte (client o altri server in caso di architettura distribuita) in una comunicazione
- **l'autorizzazione**, cioè la concessione dei diritti di accesso ad una risorsa in base all'identità della parte richiedente
- **la fiducia**, cioè la capacità di confidare nel risultato di un'operazione, come ad esempio l'autenticazione o l'autorizzazione, eseguita da terze parti
- **la riservatezza**, cioè la capacità di mantenere l'informazione privata durante la trasmissione o la memorizzazione
- **l'integrità**, cioè la capacità di prevenire che l'informazione possa essere modificata da terzi

REST API - SICUREZZA

L'approccio REST è largamente basato sul protocollo HTTP e la gestione della sicurezza ne eredita le caratteristiche ormai collaudate.

Per gestire l'identità si fa affidamento all'HTTP Basic Authentication (Base64) o HTTP Digest Authentication che con l'accoppiata al protocollo SSL critta anche le informazioni che vengono scambiate in rete. L'adozione di HTTPS garantisce la riservatezza e l'integrità nella trasmissione delle informazioni, coprendo gli altri aspetti di sicurezza.

```
GET /ordini/?123 HTTP/1.1
Host: www.mionegozio.com
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Delegare ad un servizio esterno il compito di gestire i client (OpenID e OAuth2).

E' importante associare codice di errore HTTP con gli errori che si creano, nel modo più specifico possibile.

REST API - FRAMEWORK

L'uso di un framework può semplificare l'implementazione di un Web Service, permettendoci di concentrarci sulla logica applicativa senza pensare ai dettagli relativi alla presentazione del servizio ed alla comunicazione con il client.

Un framework dovrebbe mettere a disposizione un sistema di URI Routing per mappare un URI di una risorsa a una logica da noi implementata.

Un framework REST si occupa dei dettagli della comunicazione tra client e server, separando la logica di gestione (business logic) della risorsa dagli aspetti della comunicazione. Il framework deve richiedere di specificare il codice da eseguire in corrispondenza a ciascun metodo HTTP.

Un framework prevede la standardizzazione delle risorse in formati predefiniti (Atom xml o JSON) ma consente anche l'implementazione di un formato specifico.

Un framework potrebbe mettere a disposizione funzionalità di supporto come OAuth e OpenID.

La scelta del framework è legata al linguaggio di programmazione e dalle funzionalità che offre.