Spring

1. Spring e Spring boot
2. Spring MVC
3. Spring batch
4. Spring Security

IBM Client Innovation Center
Italy

Spring

- Core
- MVC
- Batch

Presented by

Gianluca Avizzano

IBM Client Innovation Center - Napoli

Napoli, 30/05/2022

Spring: Una definizione

Che cos'è un framework?

«[…] nello sviluppo software, [un framework] è un'architettura logica di supporto […]
sul quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.» (*)

Framework e **libreria** vengono spesso confusi tra loro:

- In entrambi I casi si tratta di codice scritto da altri sviluppatori utilizzato per velocizzare I tempi di sviluppo.
- Mentre però una libreria espone delle funzioni che vengono in aiuto per un Progetto. Un framework definisce la struttura del codice del Progetto.

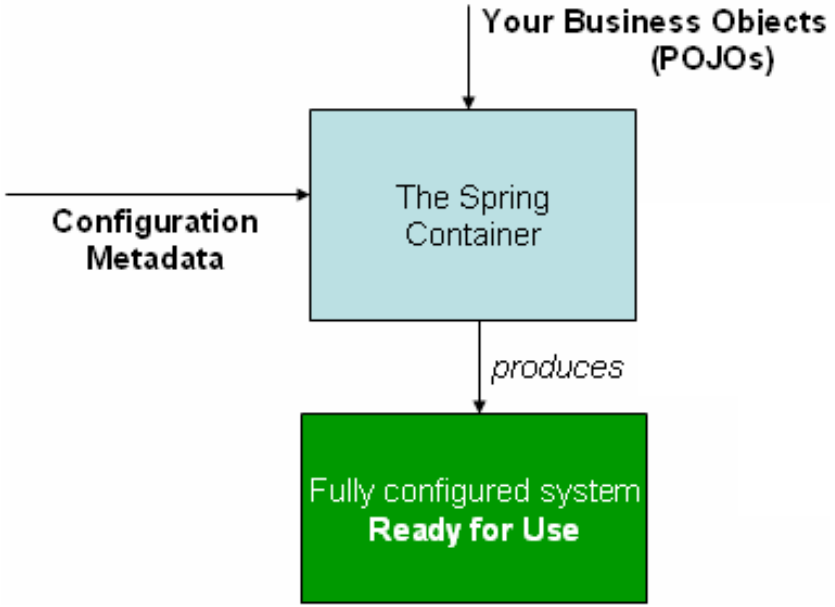
Che cos'è Spring?

Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java.

(*) Wikipedia l'enciclopedia libera, Articolo "Framework"

Spring Core

- **Bean:** Un oggetto che è istanziato, assemblato e gestito dal container di Spring.
- **Application Context:** “Interfaccia centrale che fornisce la configurazione ad un’applicazione” (*) . La configurazione può essere XML o basata su Java Annotation.
- **Dependency Injection:** è un Design Pattern in cui un componente di terze parti si occupa di istanziare le dipendenze di un’entità. Con Spring, l’implementazione di questo pattern avviene tramite l’iniezione di bean attraverso i setter delle variabili di istanza o attraverso il costruttore. Senza Spring, il programmatore deve necessariamente fornire le dipendenze definendole all’interno della classe java (tramite l’operatore new).



(*) https://docs.spring.io/spring-framework/docs/3.0.x/javadoc-api/org/springframework/context/ApplicationContext.html

Bean

Bean properties
Class
Name
Scope
Constructor arguments
Properties
Autwiring mode
Lazy initialization mode
[...]

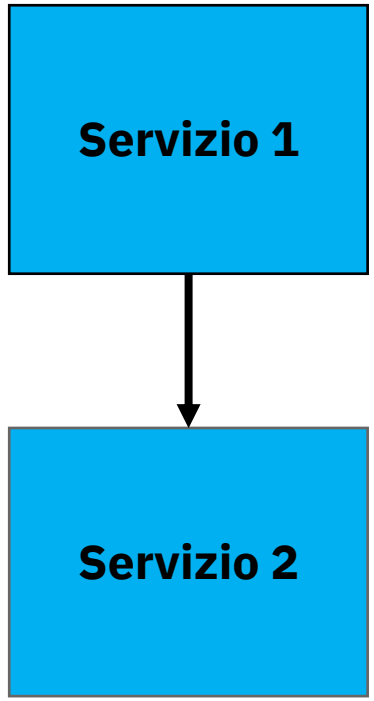
Dichiarazione di un bean tamite XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 https://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="accountDao“ class="org.springframework.samples.jpetstore.dao.jpa.JpaAccountDao">
            <!– here for example name=“itemDao” ref=“itemDao”-->
        </bean>
        <bean id="itemDao“ class="org.springframework.samples.jpetstore.dao.jpa.JpaItemDao">
       </bean>
            <!-- more bean definitions for data access objects go here -->
</beans>
```

Spring Core: Esempio di DI senza Spring

```java
public class Servizio1 {
    private String nome;

    public Servizio1(){
        this.nome = "Servizio 1";
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```java
public class Servizio2 {
    private Servizio1 servizio1;

    public Servizio2() {
        this.servizio1 = new Servizio1();
    }

    public void doSomething() {
        System.out.println("Nome servizio 1: " + servizio1.getNome());
    }

    public Servizio1 getServizio1() {
        return servizio1;
    }

    public void setServizio1(Servizio1 servizio1) {
        this.servizio1 = servizio1;
    }
}
```



Al Servizio2 è delegato l'onere di istanziare l'oggetto "servizio1".
Si consideri che in un caso reale, il Servizio2 può avere molti attributi da istanziare, e a loro volta questi attributi possono essere composti da molti altri attributi.

Spring Core: Esempio di DI con Spring

```java
@Component
public class Servizio1 {
    private String nome;

    public Servizio1(){
        this.nome = "Servizio 1";
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```java
@Component
public class Servizio2 {

    @Autowired
    private Servizio1 servizio1;

    public void doSomething() {
        System.out.println("Nome servizio 1: " + servizio1.getNome());
    }

    public Servizio1 getServizio1() {
        return servizio1;
    }

    public void setServizio1(Servizio1 servizio1) {
        this.servizio1 = servizio1;
    }
}
```

L'annotazione @Component verrà intercettata da Spring il quale istanzierà un bean per ciascun servizio. I bean avranno un identificativo che di default è il nome della classe con l'iniziale in minuscolo. Con l'annotazione @Autowired, Spring inietterà l'istanza del servizio1, pertanto istanziare tramite il costruttore non è più necessario.

Spring Core: Singleton Pattern

Il singleton è un pattern creazionale che assicura l'istanziamento di un solo oggetto di una classe, fornendo poi un punto di accesso comune a chi ne fa utilizzo.

A destra, è riportata l'implementazione di tale pattern.

I bean creati da spring sono di default dei singleton. Ad esempio, i bean "servizio1" e "servizio2" sono entrambi dei singleton, se si aggiungesse un terzo servizio che dipende dal "servizio1", l'oggetto "servizio1" iniettato nel "servizio3" sarebbe lo stesso di quello iniettato nel "servizio2".

Implementazione di un singleton senza spring.

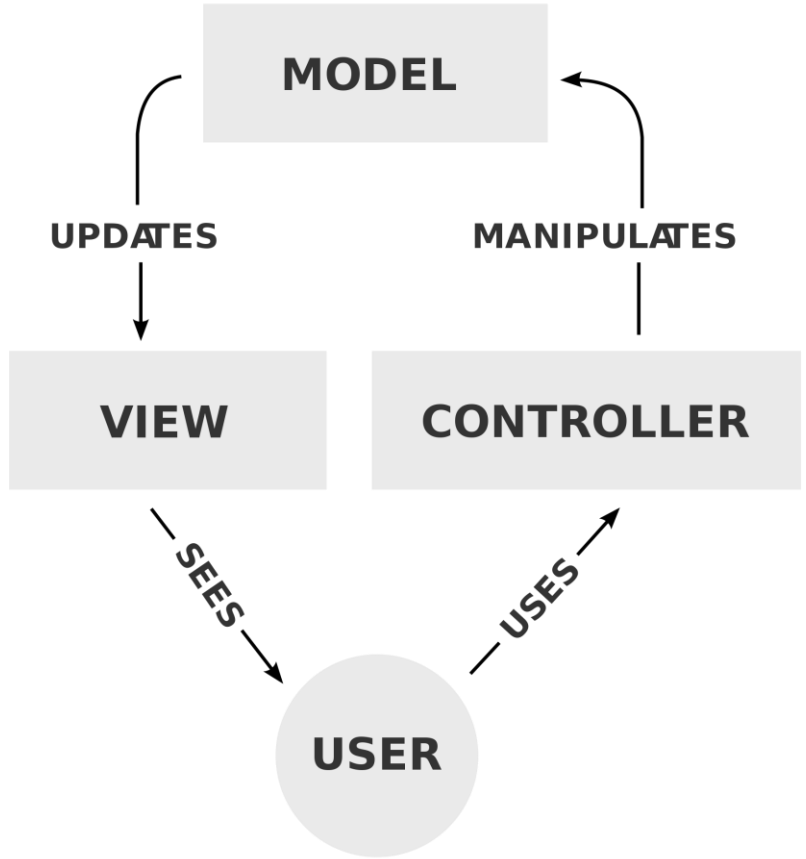
```
public class Singleton {

    private static Singleton singleton;

    private Singleton() {

    }

    public static synchronized Singleton getIstance() {
        if(singleton == null) {
            singleton = new Singleton();
        }
        return singleton;
    }

    public String getInformation() {
        return "Singleton example.";
    }

}
```

```
@RestController
@RequestMapping("/service")
public class DependencyController {

    @GetMapping(value="/singleton")
    public String getSingleton() {
        return Singleton.getIstance().getInformation();
    }
```

IBM Client Innovation Center
Italy

Model View Controller

“Model-view-Controller (MVC)” è un pattern architetturale utilizzato per sviluppare interfacce utente dividendo la logica in tre elementi interconnessi. In questo modo è possibile separare la rappresentazione interna delle informazioni dalle informazioni presentate ed accettate dall’utente. (*)



I tre componenti sono:

- Controller: Considera i dati in input dell’utente, eventualmente li convalida e li passa al Model.
- Model: Gestisce la struttura dati utilizzata poi per renderizzare contenuti personalizzati per l’utente.
- View: Si occupa di mostrare le informazioni all’utente in un determinato formato.

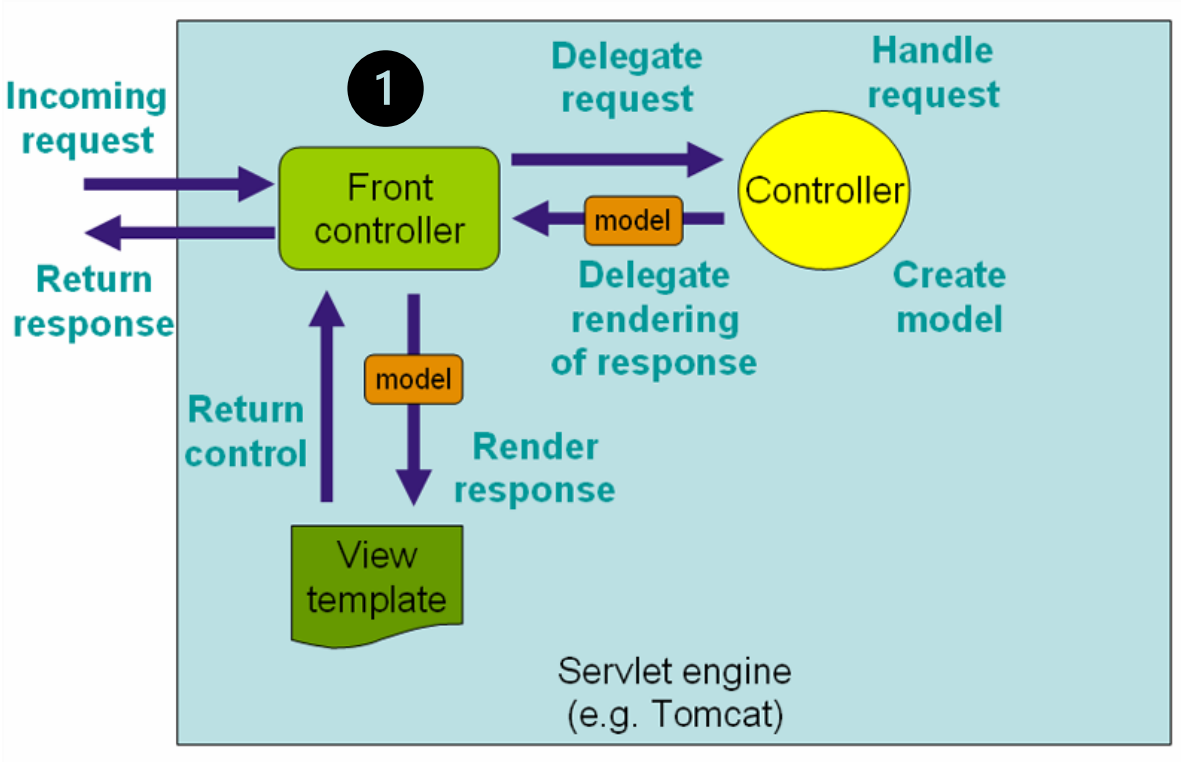
(*) Da Wikipedia

Spring MVC

Spring MVC fornisce una soluzione al pattern MVC.

Il “Front Controller” è un pattern definito come “un controller che gestisce tutte le request di un sito web”. Esso è esposto come unico servizio della web-application, e delega le request a specifiche sotto-risorse. È un pattern comune anche ad altri framework.

Con Spring MVC, il “Front Controller” prende il nome di “DispatcherServlet”.

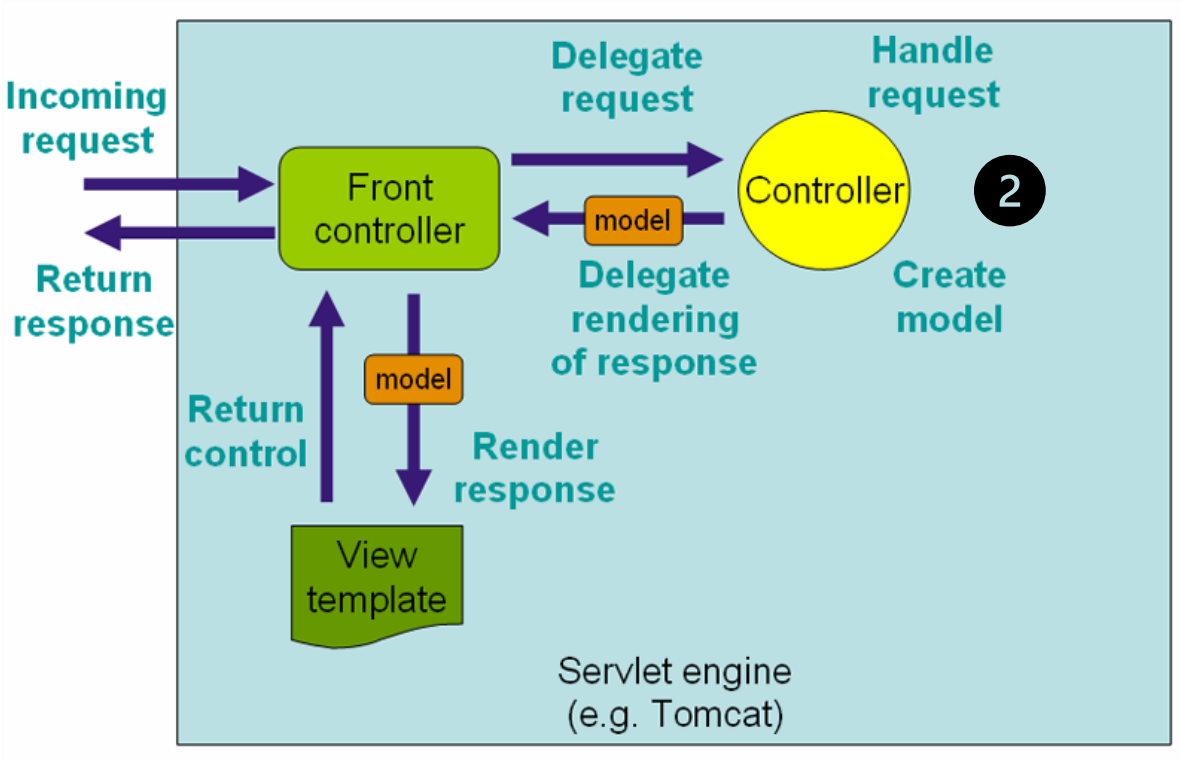


Spring MVC

Un controller è una classe Java su cui è apposta l'annotation @Controller.

I metodi del controller possono essere mappati ad un determinato URI con l'annotation @RequestMapping (Oppure @GetMapping[...])

I metodi mappati nei controller, gestiranno un oggetto "model", il model conterrà le informazioni dedicate alla request. Consiste in una mappa con chiave di tipo stringa e valore di tipo oggetto.



Controller: Esempio

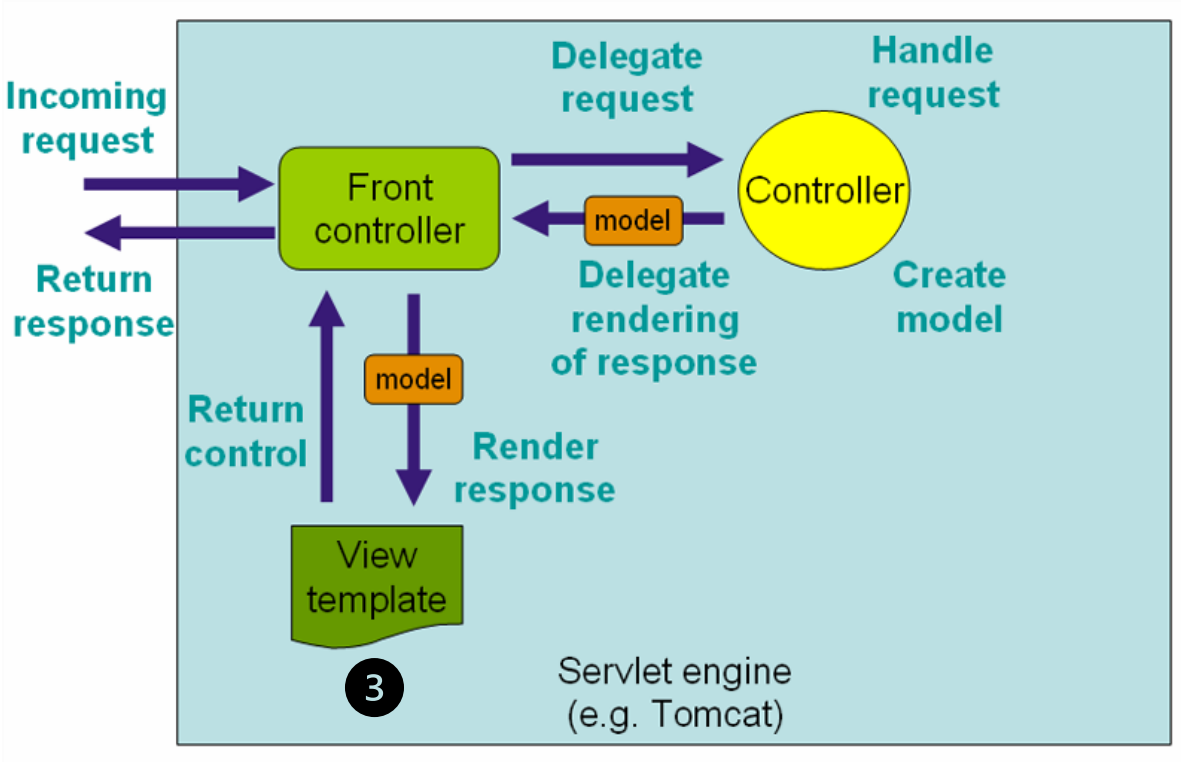
```
@Controller
@RequestMapping(value="/search")
public class SearchController {

    @Value("${searchpage.search.limit}")
    private Integer limitPagination;

    @Resource(name="productServiceDefault")
    private ProductService productService;

    @GetMapping
    public String unspecified(@RequestParam(required = false, name="line") String productLine , Model model) {
        List<Products> products = productService.getProductByLine(productLine);
        model.addAttribute("products", products);
        model.addAttribute("line", productLine);
        return "search";
    }
```

Spring MVC

Mentre il controller, dato un input, fornisce un model che contiene i dati di risposta per l'utente, il View template fornisce un modo per formattare dinamicamente i dati da presentare all'utente.

Esistono diversi template engine. Tra i più famosi: Java Server Pages (JSP), Thymeleaf, FreeMarker.

La Java Server Pages è una tecnologia che permette di combinare codice java e codice HTML col fine di produrre un contenuto dinamico in funzione dell'utente.



Spring boot

Spring boot è un tool che permette di sviluppare più velocemente web application e microservizi.

Esso fornisce una configurazione che spetterebbe altrimenti allo sviluppatore, ad esempio: se c'è Spring MVC sul classpath, è necessario fornire un servlet container. Spring boot provvede al servlet container (che di default è tomcat).

Spring initialzr è un tool che dati i moduli in input richiesti per l'applicativo che si vuole sviluppare, genera un template basato su spring-boot

Template progetto per gli esercizi

Col fine di velocizzare la fase di creazione del Progetto, si consiglia l'utilizzo di Spring Initializr, i moduli necessari sono:

- Spring Web
- Spring Data JPA
- MySQL Driver
- Spring Batch

Il pom dovrà contenere anche le dipendenze per poter utilizzare le JSP/JSTL.

Un esempio esaustivo di POM è disponibile qui.

Spring MVC: Esercizi

L’obiettivo degli esercizi, è quello di implementare un sito e-commerce relativo alla vendita dei prodotti per la base di dati oggetto di studio del modulo 2 “classicmodels”.

Esercizio 1:

Il sito e-commerce necessita di una Product List Page (PLP). La PLP consiste in una pagina web che contiene una Maschera per cercare i prodotti data la linea di prodotto. Sottomessa la maschera, la pagina si ricarica mostrando la lista di prodotti associati alla linea di prodotto in input. In particolare si vuole mostrare per ciascuno di essi: codice, nome, prezzo.

- Nel caso in cui la ricerca non fornisca alcun risultato, bisognerà informare l’utente con un messaggio.

Spring MVC: Esercizi

Esercizio 2:

Il sito e-commerce necessita inoltre di una Home Page, essa conterrà:

- Un link che indirizza alla pagina di ricerca del punto 1,
- Successivamente, verranno proposti i primi dieci prodotti (Codice, nome, descrizione), ottenuti interrogando il database (l'ordine non è importante).
- A piè dei dieci prodotti, è necessario aggiungere un link "Mostra di più", il link porterà ad una pagina contenente i primi 50 prodotti ordinati per codice.
    - La lista sarà paginata: conterrà un link alla pagina 2 che mostrerà i successivi 50 prodotti.
    - La pagina conterrà un link "Torna indietro" per tornare alla Home Page

Spring batch

Un batch è un programma assegnato ad un computer per essere eseguito senza ulteriori interazione con l'utente.

In generale, un batch si occupa di leggere da un database o da un file (in gergo "flusso") dei dati (fase di reading), effettuare poi delle operazioni (fase di processing), e scrivere i risultati su un file di output e/o aggiornare la base di dati (fase di writing).

Spring batch è un modulo di Spring che fornisce una serie di entità utilizzabili ed estendibili per poter implementare un batch.

Terminologia:

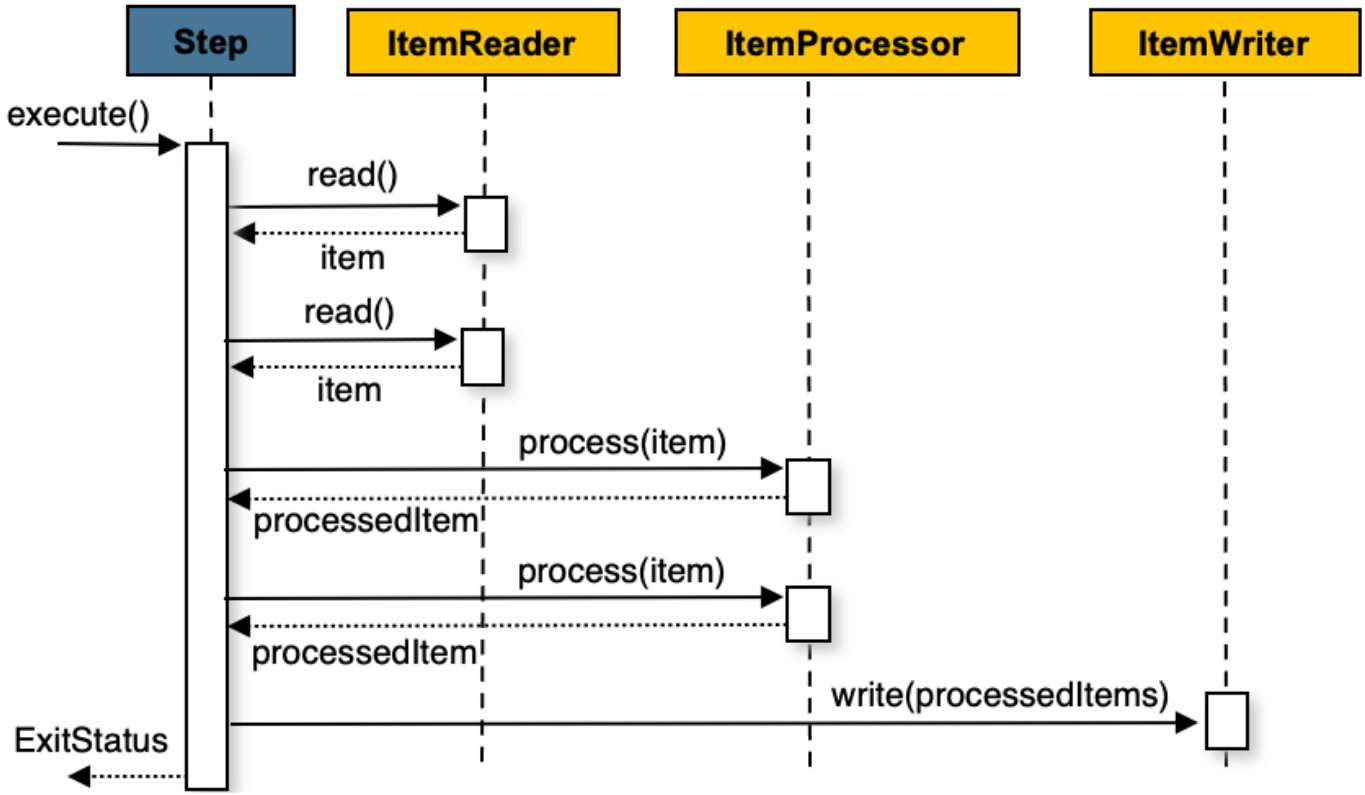
- **Job**: Incapsula l'intero processo del batch.
- **JobInstance**: rappresenta concettualmente un esecuzione di un job. Ad esempio se un job è eseguito il giorno 1, e durante l'esecuzione viene interrotto per poi essere ripreso il giorno 2, l'istanza è sempre una, mentre le esecuzioni sono state due. Per questo esiste un'associazione uno a molti tra JobIstance e JobExecution.
- **JobExecution**: Un'esecuzione corrisponde ad un nuovo avvio del batch.
- **JobParameter**: Sono parametri accedibili da codice, tramite I parametri possiamo ad esempio distinguere un esecuzione da un'altra (a parità di istanza)
- **Step**: Un job è composto da N step, con gli step è possible suddividere il batch in fasi sequenziali.
- **ExecutionContext**: Una mappa chiave-valore, contestualizzata a livello di step o di job, possiamo quindi ad esempio salvare in memoria degli oggetti che possono servirci negli step successivi.

Configurazione di uno step

Ogni step implementa le seguenti interfacce:

- **ItemReader**: Il metodo *read()* si occupa di fornire un item da processare, lo step si conclude quando non ci sono più item da processare (l'item di output è *null*).
- **ItemProcessor**: Il metodo *process(I item)* si occupa di trasformare l'oggetto di output del reader (Può arricchire l'oggetto, o scartarlo se necessario o ancora può restituire un oggetto di un altro tipo).
- **ItemWriter**: Il metodo *write(List<I> items)* scrive su file/database un chunk di items processati dal processor.

La logica di business verrà inserita all'interno delle implementazioni di queste interface.

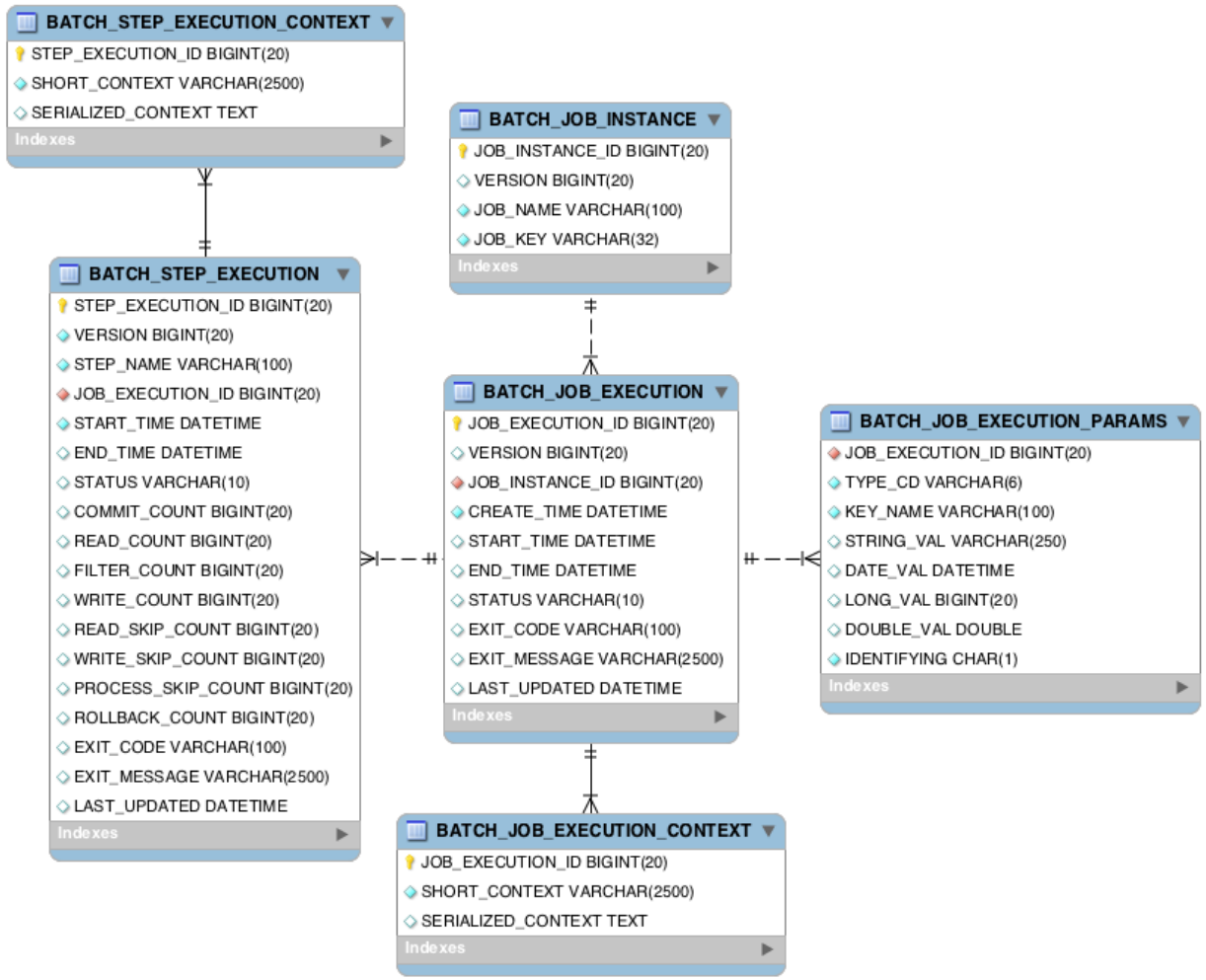


Spring batch:
Meta-data schema

Col fine di eseguire analisi, ricavare metriche, o effettuare sessioni di debug , è utile sapere che Spring mette a disposizione “Out of the Box” delle tabelle che contengono meta-dati.

È possibile ad esempio verificare :

- Quali e quante sono state le esecuzioni di un job
- Quali sono stati i tempi di esecuzione
- Se le esecuzioni sono andate a buon fine.



Spring batch: Esercizio

Un ETL, integrato con altri componenti, si occupa di raccogliere e processare i dati relativi ai prodotti, generando poi un file CSV contenente una lista di essi. I prodotti possono essere nuovi (non ancora presenti all'interno del sistema informativo), oppure già esistenti, su cui si intende solo aggiornare alcuni campi. In entrambi i casi, il CSV fornito, contiene tutte le informazioni del prodotto.

Si vuole implementare un automatismo che si occupi di aggiornare il sistema informativo processando il file CSV ricevuto in input.

Un esempio di file CSV è presente qui.

Experience.
Create.
Inspire.

Gianluca Avizzano

IBM Client Innovation Center
Italy