Introducción a Spring Batch

Java

Es un lenguaje de programación multiplataforma y multiparadigma donde se destaca el orientado a objetos.

Spring Boot

Es una herramienta que facilita el desarrollo de aplicaciones web y microservicios.

Se basa en tres principios:

1. Configuración automática.
2. Enfoque obstinado de la configuración. Spring facilita todo lo que necesitas en una sola interfaz.
3. Capacidad de crear aplicaciones independientes. Sólo te debes enfocar en escribir el código y Spring hace el resto.

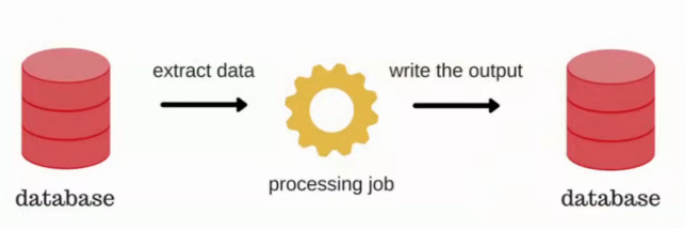
¿Qué se puede hacer con Spring Boot?

- Microservices: facilita el mantenimiento y escritura del código teniendo aplicaciones más pequeñas, automatizadas y listas para ejecutarse.
- Reactive: aplicaciones asíncronas que no son bloqueantes (permiten más llamadas de las que pueden manejar a través de hilos).
- Cloud: se pueden hacer sistemas distribuidos y trasladarlos a la nube en conjunto con aplicaciones y configuraciones.
- Web apps: facilita la configuración para hacer APIs RESTful y sistemas bidireccionales basados en eventos.
- Event driven apps: reflejan el funcionamiento al cambio.
- Serverless: el usuario se enfoca en realizar la lógica de negocio y el sistema proveedor se ocupe de todo lo demás.
- Spring Batch: se describirá más adelante su funcionamiento.

Batch Processing

El procesamiento por lotes (*batch processing*) es un método para ejecutar "trabajos" repetitivos de alto volumen de datos con poca o ninguna interacción del usuario.

En resumen, se agarra información de un lado, transformarla a otro formato y pasarla a otro lado.



Es necesario si se necesita procesar mucha información frecuentemente para evitar desbordamientos de memoria.

Conceptos básicos del Batch Processing

Los parámetros esenciales de este método buscan responder el quién, qué, dónde y cuándo. De manera más específica:

- Quién está enviando el trabajo.
- Qué programa lo ejecutará.
- La ubicación de las entradas y salidas.
- Cuándo se debe ejecutar el trabajo.

¿Los sistemas nunca fallan?

A pesar de ser muy robustos, jamás se asegura que nunca van a fallar, a pesar de que la probabilidad de que se caiga sea menor a 1%, por lo cual siempre necesitaremos de métodos de contingencia.

Si llegara a haber un fallo en el microservicio que impida comunicarse con un sistema, la información enviada se guarda en un respaldo para no perderla y, cuando se pueda restablecer una conexión, mandarla sin tener pérdidas.

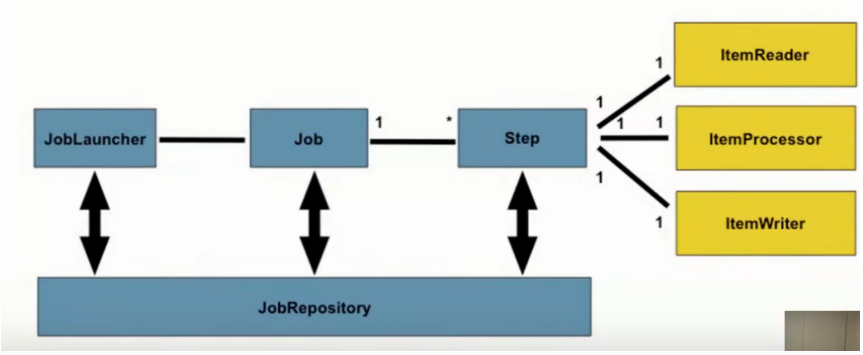
Spring Batch

Es un framework (curiosamente dentro de otro framework) enfocado específicamente en la creación de *batch processing*.

Proporciona una gran cantidad de componentes que intentan dar soporte a las diferentes necesidades que surgen a la hora de crear estos programas.

Si se tiene una entrada de muchísimos datos, en vez de leerlos con nuestros recursos de hardware (lo cual puede causar un desbordamiento si no se tiene la infraestructura adecuada), Spring Batch puede hacer que se lean estos registros de cinco en cinco hasta leer todo con recursos del servidor; esto se logra en específico con el *chunk oriented processing*, es decir, se lee la información en bloques.

Componentes principales de Spring Batch



- `Job`: es un proceso que se ejecutará en un tiempo determinado. También es un contenedor de `Steps` y debe contener al menos uno de éstos.
- `Step`: es un elemento independiente se encuentran dentro de `Job` como unidades. Representa una de las fases de la que está compuesto el `Job`.
- `JobRepository`: guarda la información de los `Job` y de los `Step` y nos da retroalimentación; ya sea si no ocurrió ningún error o si ocurrió uno, en qué momento y qué sucedió. En caso de encontrar un error, a Spring Batch se le dice cómo solventarlo en caso ya tenerlo contemplado.
- `JobLauncher`: ejecuta los `Job` dentro del contexto de Spring suministrando los parámetros de entrada deseados.

Los `Step` constan de tres partes:

- `ItemReader`: lee los registros recibidos en la entrada como se mencionó anteriormente, en bloques que ocupen menos recursos.
- `ItemProcessor`: se encarga de procesar los datos recibidos y comprobar que estén en el formato deseado, sino, actuar en consecuencia como se mencionó anteriormente. En esta parte se encuentra la lógica del negocio.
- `ItemWritter`: esta parte la consideramos como la salida y la escritura.

Aunque también puede tener únicamente el `ItemProcessor`. Es el caso del tasklet con el código que se desea ejecutar en el `Step`.

Ejemplo de Spring Batch

Imaginemos una aplicación de Spring Boot con Spring Batch que realiza el siguiente proceso batch:

- La lectura de un fichero csv.
- El procesamiento del fichero.
- Escritura en una base de datos h2.

En el `ItemReader` recibe la entrada para después pasarla al procesador. En el procesador se implementa la lógica de negocio que se requiera. Finalmente, en el `ItemWritter` se guarda lo que regresa el procesador en un repositorio.

En el método `processJob()` se define lo que queremos que haga éste. Además de llamar al *listener* que nos notifique antes y después de que comenzara el `Job`, lo cual lo observa el usuario con un `log`. Los `Job` tienen estados que indica si completaron su proceso. dentro del `processJob()` se encuentra el método `flow(step())`, donde en el `step()` se puede implementar la forma en la que se leen los datos y cuántos se leen.

Con ayuda de los *listeners*, podemos notificar qué datos se leyeron, si ya se leyó algo o si hubo algún error. Éstos se pueden implementar antes y después de la lectura y escritura, para así tener un control de todas las fases (saber qué se leyó, qué escribió o qué hará).

Finalmente, en la tabla h2 se puede observar la información acerca de los `Job`, como la información que se mencionó anteriormente.