



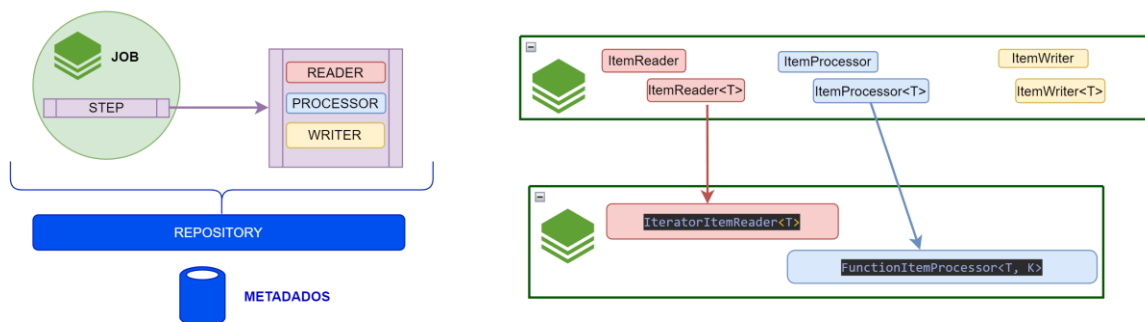
## os chunks do Spring Batch?

Existem dois tipos de steps no Spring batch, os **Tasklet** e os **Chunk**. **Tasklet** são steps simples, que podem ser feitos de uma vez sem maior complexidade, por exemplo, criar um diretório.

**Chunks** são steps que são feitos em **pedaços**, ideais para tarefas mais complexas e com um volume maior de dados onde a integridade é importante. A estrutura de um **chunk** é composta por um **leitor**, um **processador(opcional)** e um **escritor**.

Em uma visão geral de execução de um batch nós **temos o job** que tem o **step** e o **step se for um chunk** tem sua estrutura. Todos rodando sobre o **repositório (metadados do Spring Batch salvos em banco)** para manter consistência.

O **leitor**, **processador** e **escritor** são provenientes de uma Interface, então o Spring fornece implementações já prontas desses componentes, e se necessário, a possibilidade de implementação mais específica nossa:



## implementando um chunk

A **implementação do job** se mantém **inalterada**, ele recebe e **starta o(s) Step**. Agora no **Step** é que tem a maior diferença, dizemos o tipo do **step**(*nesse caso um chunk que recebe um inteiro e devolve uma String*) e passamos o **leitor**, **processador** e **escritor**:

```
@EnableBatchProcessing
@Configuration
public class ChunkBatchBase {

    private final JobBuilderFactory jobBuilderFactory;

    public ChunkBatchBase(JobBuilderFactory jobBuilderFactory) {
        this.jobBuilderFactory = jobBuilderFactory;
    }

    @Bean
    public Job printEvenOrOdd(Step evenOrOddStep) {
        return jobBuilderFactory
            .get("evenOrOddJob")
            .start(evenOrOddStep)
            .incrementer(new RunIdIncrementer())
            .build();
    }
}

@Configuration
public class EvenOrOddStepConfig {

    private final StepBuilderFactory stepBuilderFactory;

    public EvenOrOddStepConfig(StepBuilderFactory stepBuilderFactory) { this.stepBuilderFactory = stepBuilderFactory; }

    @Bean
    public Step evenOrOddStep() {
        return stepBuilderFactory
            .get("evenOrOddStep")
            .<Integer, String>chunk(1)
            .reader(countUntilTenReader())
            .processor(evenOrOddProcessor())
            .writer(printWriter())
            .build();
    }

    public IteratorItemReader<Integer> countUntilTenReader() {
        List<Integer> numbersOneToTen = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        return new IteratorItemReader<Integer>(numbersOneToTen.iterator());
    }

    public FunctionItemProcessor<Integer, String> evenOrOddProcessor() {
        return new FunctionItemProcessor<Integer, String>() {
            @Override
            public String process(Integer item) {
                return item % 2 == 0 ? String.format("%s is Even", item) : String.format("%s is Odd", item);
            }
        };
    }

    public ItemWriter<String> printWriter() {
        return items -> items.forEach(System.out::println);
    }
}
```



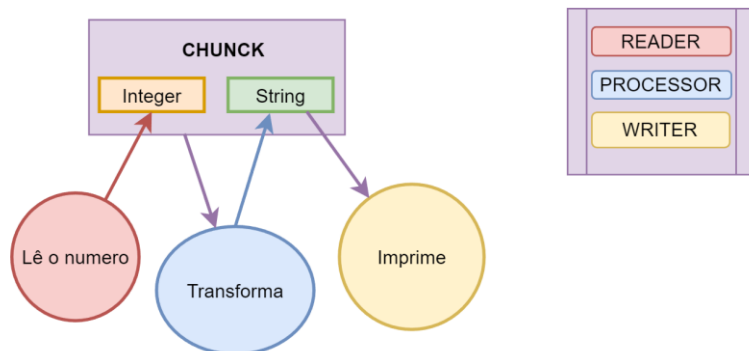
## Etapas do chunk

O **leitor** está utilizando uma implementação pronta do **Spring Batch**, chamada de **IteratorItemReader**, nesse caso eu to pedindo pra ele iterar sobre uma lista de 1 a 10, ou seja, meu chunk vai ter como entrada 10 números fornecidos pelo leitor.

O **processador** também está utilizando uma implementação pronta do **Spring Batch**, chamada **FunctionItemProcessor**, já que o processamento vai consistir em dizer se um número é par ou ímpar, ele pode ser uma função simples. Nesse caso eu pego o **inteiro passado pelo leitor**, **processo e retorno uma String**.

Já o **Writer** não está usando nenhuma implementação específica, estou apenas imprimindo o resultado do processamento.

Resumindo, dizemos ao **chunk** o que vai **entrar(Integer)** e o que vai **sair(String)**, o **Reader entrega a entrada** ao **chunk**, o **processador transforma a entrada** e o **writer o joga na tela (NESSE CASO)**:



Sem dúvida, um dos elementos mais importantes desse processo todo é o **tamanho** do **chunk**, o **número passado como argumento** ao construir o **chunk**:

```
@Bean
public Step evenOrOddStep() {
    return stepBuilderFactory
        .get("evenOrOddStep")
        .stepBuilder()
        .<Integer, String>chunk( chunkSize: 1)
        .reader(countUntilTenReader())
        .writer(countUntilTenWriter())
        .build();
}
```

Esse número dita o **intervalo de commit do chunk**, basicamente falando: **“De quanto em quanto dessas entradas eu vou commitar por vez?”**, nesse caso, **temos 10 números, 10 entradas**. Essas 10 entradas **vão ser lidas, processadas e escritas de 1 em 1**.

Transições e commits pesam no banco de dados, então talvez faça sentido que eu apenas comite depois de fazer o batch das 10 entradas, eu poderia fazer isso. Mas o ideal é sempre levar em consideração um balanço de quanto recurso a máquina rodando o batch pode fornecer, e o quanto você quer comitar por vez. Esse tipo de balanceamento é totalmente empírico, mas muito importante de ser pensado.