Maven Dependency Analysis using Akka Streams

Nikita Servais

December 13, 2023

Akka Streams for Concurrent Processing

Akka Streams were chosen as the foundation for this project due to their robust framework for building streaming applications with built-in support for backpressure. This ensures efficient resource utilization without overwhelming system components, which is vital for processing a large dataset of Maven library dependencies.

Scala for Functional and Object-Oriented Paradigms

Scala offers a seamless integration of functional and object-oriented programming paradigms, ideal for creating immutable data structures and employing functional transformations. This aligns well with Akka Streams, which is built on top of Scala and Akka actors.

Data representation

The MavenLibrary class represents a Maven library with its dependencies. The ParsedDependency represents a CSV line. The Library type is a groupID, artifactID and a version. The dependency type is a GroupID, artifactID, version and a dependency type.

Akka Stream Components

Source

FileIO.fromPath is used for reading the CSV file, offering non-blocking IO operations essential for stream processing.

parseFile

Parses the CSV file, using `Framing.delimiter` for efficient line handling.

instantiateClasses

Transforms CSV lines into *ParsedDependency* objects.

groupByLibraryName

Groups dependencies by library, showcasing stateful streaming operations. Using groupBy, the stream is split into substreams, one for each library. Then fold the substream into a single value, the library and its dependencies, MavenLibrary. Finally, merge the substreams back into a single stream. It takes the max number substreams, this is to limit the number of entries, in this case the number of libraries.

throttleGroups

Throttles the stream to groupsPerSecond argument every second (in this case 10 libraries per second).

bufferGroups

Buffers the stream to bufferGroups argument (in this case 100 libraries). The strategy used is OverflowStrategy.backpressure, this is to ensure that the stream does not overflow.

countDependenciesFlow

This is the main flow of the application, it counts the number of dependencies for each library. It makes use of GraphDSL to create a custom flow. One Graph is to balance the stream, this is to ensure that the stream is evenly distributed between the counters. A second Graph to count the dependencies types for each library. The count graph, count the two types of dependencies, compile and runtime, separately. This count is added to the MavenLibray object with field names compileDependencies and runtimeDependencies. Finally, the two counts are merged into a single stream.

Sink

Sink is used for outputting the final result, demonstrating the role of a sink in stream processing. It use a custom *toString* method to output the result in a special format.

Conclusion

The Maven Dependency Analysis project demonstrates the capabilities of Akka Streams in processing large datasets. By leveraging Scala and Akka Streams, the application efficiently and effectively analyzes Maven library dependencies.