Batch and Stream Processing: Realtime Analysis of Big Data

Marcel Stolin
marcelpascal.stolin@studenti.unitn.it

Abstract

Since the beginning of Big Data, batch processing was the most popular choice for processing large amounts of generated data. These existing processing technologies are not suitable to process the large amount of data we face today. Research works developed a variety of technologies that focus on stream processing. Stream processing technologies bring significant performance improvements and new opportunities to handle Big Data. In this paper, we discuss the differences of batch and stream processing and we explore existing batch and stream processing technologies. We also explain the new possibilities that stream processing make possible.

1 Part 1

1.1 Introduction

In the second lecture of the course, an implementation of a basic HTTP server with the name *TinyHttpd* was introduced. The functionality of the HTTP server is limited to opening `.html` files and delivering the content via the HTTP 1.1 protocol to the client.

The task of part 1 of the first assignment, is to extend the *TinyHttpd* implementation by implementing the functionality to launch an external process. Therefore, the client sends a request via the URL `http://localhost:8000/process/reverse?par1=ROMA`. Then, the server executes an external Java process, which reverses the string `ROMA`, given in the query `?par1=ROMA`, and responses the result of the external Java process to the client via HTTP 1.1.

Given the above mentioned task description, the following steps have to be implemented:

1. Create a Java application, called *StringReverser*, which takes a valid String as input and returns the reversed string

2. Extend the *TinyHttpd* implementation to launch external processes when requested by client via the URL `http://localhost:8000/process/PROCESS_NAME?PROCESS_PARAMETERS`

1.2 Conceptual Design

Given the problem statement introduced in Section 1.1, a new application called *StringReverser* needs to be implemented, and the *TinyHttpd* server has to be extended in a way to launch an external Java process.

1.2.1 StringReverser

The *StringReverser* application is a simple terminal application. It takes any valid String as an input and returns the reversed String as the output. It can be executed via the console, for example the command `$ java -jar StringReverser.jar ROMA` should responses the string *AMOR*.

1.2.2 TinyHttpd

Whenever the client makes a request via the URL `http://localhost:8000/process/PROCESS_NAME?PROCESS_P` the server is supposed to start an external Java process, waits for the output of the process, and responses the process output to the user. The client has the possibilities to specify which process has to be executed. Given the URL `http://localhost:8000/process/reverse?par1=ROMA`, the user explicitly requests to launch the *reverse* process with the given query `par1=ROMA` as the process input. It is important to mention, that each process takes individual parameters as input. For the above mentioned `StringReverser`, only the value of the first parameter in the given query is important. All other parameters, and the parameter key, are therefore ignored.

1.3 Implementation

To implement given conceptual design, The Java programming language is used in version OpenJDK 17[1].

1.3.1 StringReverser

The application *StringReverser* is a simple Java project. It is composed of a single Java class called `StringReverser` as shown in Figure 1. The source code is shown in Figure 2, and it consists of a `main` method and a method called `reverseString`. The main method will be executed, when the application is launched via the terminal. Additionally, it checks if a String is been given as input, and if the input is valid. Otherwise, it will return an error message and exists with system code 0. If the input is a valid string, it will call the `reverseString` method, and returns the result as the output. The `reverseMethod` is responsible to reverse the given input. Therefore, it uses the `StringBuilder`[2] class to reverse the String.

[1] JDK 17 - `https://openjdk.java.net/projects/jdk/17/` (Accessed: 02/10/2021)

[2] StringBuilder - `https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html` (Accessed: 02/10/2021)

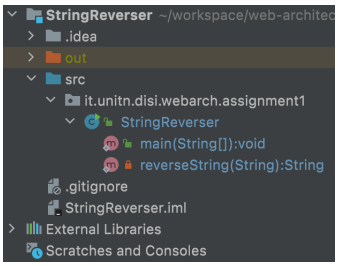

Figure 1: Project structure of the `StringReverser` application

Figure 3 shows the execution of the *StringReverser* application. The `.jar` artifact was created using the IntelliJ IDEA[3]. If the input is invalid, the `StringReverser` will print an error message to the terminal and exit with system code 0, as shown in Figure 4.

1.3.2 TinyHttpd

The foundation of the TinyHttpd project is provided by the Professor. It has to be extended to launch external Java processes, and deliver the process output to the client. As mentioned in SEC DESIGN, the client sends a HTTP request via the URL `http://localhost:8000/process/PROCESS_NAME?PROCESS_PARAMETERS`. Currently, the implementation responses with the content of the requested HTML file, if available. Otherwise, an 404 HTTP response is send to the client.

To be able to detect, if the client wants to launch an external Java process, the request has to be parsed accordingly. Therefore, the first step is to extend the existing code base with a `RequestParser` class. The `RequestParser` class is able to parse a HTTP request into its parts. Given the request `GET /process/reverse?param=roma HTTP/1.1`, it is composed of the HTTP method (`GET`), the path (`/process/reverse?param=roma`), and the HTTP protocol version (`HTTP/1.1`). Furthermore, the path has an additional query attached (?param=roma). FIG XY shows the implementation of the `RequestParser` class. After the `RequestParser` has successfully parsed the clients request, it is possible to check, via an `if` statement, if the user requested the path */request/PROCESS_NAME*. If yes, the server executes the requested process, and sends a response accordingly. Otherwise, if the client has not requested to perform a Java process, the server tries to open the HTML file according to the given path and responses the HTML content to the client. The implementation of this procedure is shown in Figure 5.

If the client has send a valid request for a process, the next step is to generate the command to launch the requested process. To keep the TinyHttpd implementation extensible, a new class called CommandFactory is added to the project, which is implemented using the Factory pattern.

[3]Create your first Java application - `https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html` (Accessed: 02/10/2021)

```java
package it.unitn.disi.webarch.assignment1;

public class StringReverser {

    public static void main(String[] args) {
        if (args.length >= 1) {
            String text = args[0];

            if (text != null) {
                String reversedString = reverseString(text);
                System.out.println(reversedString);
            }
        } else {
            System.out.println("A string to reverse is required.");
            System.exit( status: 0);
        }
    }

    /**
     * This method reverses the given string.
     *
     * @param text
     * @return Reversed text
     */
    private static String reverseString(String text) {
        StringBuilder stringBuilder = new StringBuilder(text);
        stringBuilder.reverse();
        return stringBuilder.toString();
    }

}
```

Figure 2: Implementation of the `StringReverser` class

The CommandFactory class is responsible to generate the command for the given request. It has a public static method called generateCommand, which takes the requested process name and the query of the request path as arguments. According to the given process name, it generate the command to execute the .jar artifact with the given query as input parameter. As example for the given path `/process/reverse?param=roma`, the generated command is `java -jar /Users/marcel/workspace/web-architectures/assignment_1/MiniHTTPD/jars/StringRevers "roma"`. In addition, the CommandFactory is also responsible for check if the requested process is available and if the given query is a valid parameter for the process. If not, it will throw an exception.

Next, after the command has been generated it can be performed using the ProcessBuilder class. FIG XY shows the executing of the requested process

```
~/workspace/web-architectures/assignment_1/StringReverser/out/artifacts/StringReverser_jar main*
❱ java -jar StringReverser.jar ROMA
AMOR
```

Figure 3: Successful execution of the *StringReverser* application

```
~/workspace/web-architectures/assignment_1/StringReverser/out/artifacts/StringReverser_jar main*
❱ java -jar StringReverser.jar
A string to reverse is required.
```

Figure 4: Execution of the *StringReverser* application without an input

using ProcessBuilder. It is important to mention, that the result has to be saved as string instead of printing it directly to the output stream. The reason is, because the HTTP header of the server response needs the length of the process output.

During the procedure of handling the path, generating the requested process command, and executing the process command, the following error cases can appear:

- The process path is invalid (process name missing or no process with this name exists)
- The query is invalid (query is missing or the given parameters are invalid)

If any of these cases appear, the server will response with an 400 Bad Request HTTP response.

1.4 Conclusion

2 Part 2

2.1 Introduction

2.2 Design

2.3 Implementation

2.4 Conclusion

```
// Parse the request
RequestParser requestParser = new RequestParser(req);
String method = requestParser.getMethod();
System.out.println("Method: " + method);
System.out.println("Protocol: " + requestParser.getProtocol());
String path = requestParser.getPath();
System.out.println("Path: " + path);
String query = requestParser.getQueryString();
System.out.println("Query: " + query);

if (method.equals("GET")) {
    StringTokenizer pathTokenizer = new StringTokenizer(path, delim: "/");
    // Check if client requested a java process
    if (pathTokenizer.hasMoreTokens() && pathTokenizer.nextToken().equals("process")) {...} else {
        // Otherwise, try to open the requested HTML file
        if (path.endsWith("/")) {
            path = path + "index.html";
        }
        try {
            this.sendFileResponse(path);
        } catch (FileNotFoundException e) {
            this.sendErrorResponseHeader("404 Not Found");
            System.out.println("404 Not Found: " + path);
        }
    }
```

Figure 5: Execution of the *StringReverser* application without an input