Web Architectures - First assignment

Claudio Facchinetti
<claudio.facchinetti@studenti.unitn.it>

Abstract

The report describes the process of development of the first assignment for the course "Web Architectures" held at the University of Trento in the academic year 2021/2022.

In particoular it is described the problem, the technical solutions and also some extra features that were added to the project in order to experiment in first person how it is possible to create dynamic websites without using any existing web server or templating technique / language, such as PHP or JSP.

1 Part one: creating dynamic websites

1.1 Introduction

During the lecture we were given the code to create a simple and minimal HTTP server which is only capable of receiving GET requests and responding with the content of static file requested. The first part of the assignment requested to start from this server and add to it the possibility of creating dynamic pages using a mechanism which is more or less equal to the "cgi-bin" one.
In particular what was asked was to extend the basic web server and make it able to start a new process whenever a file inside the `"process"` folder is requested. The specific request is that "`http://<machineURL>/process/reverse?par1=roma`" should start a new Java program in an external process which had to return the value of `par1` reversed, "amor" in this case.

1.2 Solution

The assignment stated that we could have started from the project explained during class time by the professor, however in this case it has been rewritten from scratch in order to have a better understanding of how to interact directly with the HTTP protocol.
The code that has been produced had the very same features as the one proposed by the professor, with the only difference that it is extensible, leaving the opportunity to add support for more HTTP methods and much more.
The project, named `TinyHTTPd`, is composed by mainly four components:

- **TinyHTTPRequest**: a class representing any HTTP request; it is able to parse the request coming from the socket in order to extract from it all the components, such as the path, the protocol version, the method, the headers and query string parameters.
- **TinyHTTPResponse**: a class representing any HTTP response; it is the class that represents any HTTP response and it can automatically encode and print itself on a print writer being a valid HTTP response.
- **Handler**: it is an abstract class which is responsible of handling TinyHTTPRequests and producing TinyHTTPResponses; the main idea is that there will be a concrete class for each HTTP method, even though this is not mandatory and in this case only the GET one has been implemented.
- **TinyHTTPConnection**: it is the class that created by the main class upon a connection is received; it is responsible of dispatching requests to the correct handler in order to get the response and make it print on the stream.

With respect to the assignment requirement the GET handler has been modified implementing the following logic: if the requested file is in the "process" folder, then start a new process (using the ProcessBuilder API) passing to it all the parsed query string parameters.
One note that could be useful is that while parsing the path the code automatically removes any `..`, so that particoular attacks known as "path traversal" can be no longer applied.

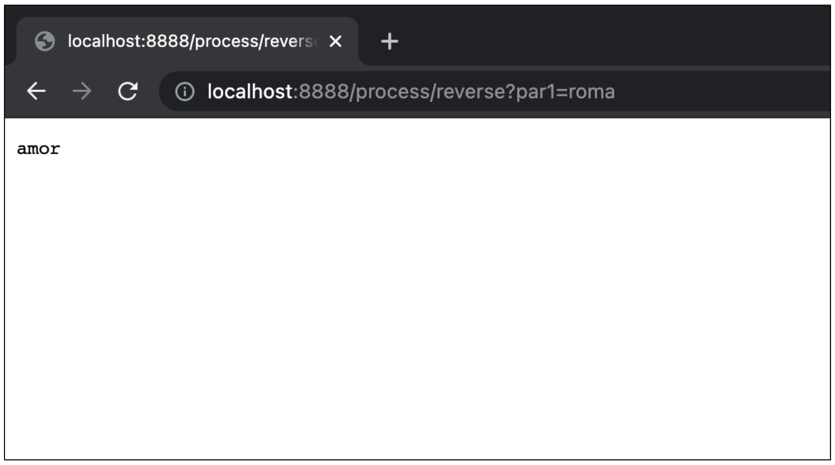
```
@Override
public TinyHTTPResponse handleRequest(TinyHTTPRequest request)
    throws IOException,
    UndefinedNameException, ClassNotFoundException,
    InterruptedException{

    if( request.getPath().startsWith("/process") ){
        return this.handleProcessRequest(request);
    } else{
        return this.handleFileRequest(request);
    }
}

private TinyHTTPResponse handleProcessRequest(TinyHTTPRequest request)
    throws IOException, InterruptedException {

    File executableFile = new File(request.getPath().substring(1) + ".java");
    List<String> cmds = new ArrayList<>();
    cmds.add("java");
    cmds.add(executableFile.getName());

    for (Map.Entry<String, String> param :
            request.getQueryStringParams().entrySet()) {
        cmds.add("-" + param.getKey() + "=" + param.getValue());
    }

    ProcessBuilder pb = new ProcessBuilder(cmds);
    pb.directory(executableFile.getParentFile().getAbsoluteFile());
    pb.redirectErrorStream(true);
    Process p = pb.start();

    BufferedReader reader =
        new BufferedReader(new InputStreamReader(p.getInputStream()));
    StringBuilder sb = new StringBuilder();
    String line;

    while( (line = reader.readLine()) != null ){
        sb.append(line).append("\n");
    }
    p.waitFor();

    String result = sb.toString();
    Map<String, String> headers = RequestHandler.getPlainTextHeaders(result);
    return new TinyHTTPResponse(request.getVersion(),
        HttpStatus.OK, headers, result.getBytes(StandardCharsets.UTF_8));
}
```

Snippet 1: Code used to start the new process depending on URL

As you can see the code is straight forward: one peculiar thing is that the error stream is redirected; this has been done because normally program write errors on `stderr` while the output on `stdout` and the desired behaviour is that we return something to the user.

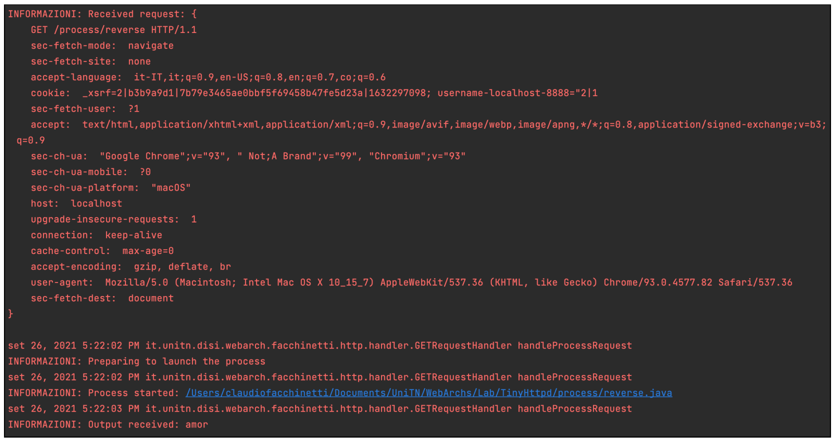
1.3 Application running

Here there are some screenshots of the application running: it is reported the screenshot of the browser as well as the logs produced by the TinyHTTPd for that request.


localhost:8888/process/revers
localhost:8888/process/reverse?par1=roma

```
amor
```

Figure 1: Result of calling the URL on Chrome



```
INFORMAZIONI: Received request: {
    GET /process/reverse HTTP/1.1
    sec-fetch-mode:  navigate
    sec-fetch-site:  none
    accept-language:  it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7,co;q=0.6
    cookie:  _xsrf=2|b3b9a9d1|7b79e3465ae0bbf5f69458b47fe5d23a|1632297098; username-localhost-8888="2|1
    sec-fetch-user:  ?1
    accept:  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
 q=0.9
    sec-ch-ua:  "Google Chrome";v="93", " Not;A Brand";v="99", "Chromium";v="93"
    sec-ch-ua-mobile:  ?0
    sec-ch-ua-platform:  "macOS"
    host:  localhost
    upgrade-insecure-requests:  1
    connection:  keep-alive
    cache-control:  max-age=0
    accept-encoding:  gzip, deflate, br
    user-agent:  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36
    sec-fetch-dest:  document
}

set 26, 2021 5:22:02 PM it.unitn.disi.webarch.facchinetti.http.handler.GETRequestHandler handleProcessRequest
INFORMAZIONI: Preparing to launch the process
set 26, 2021 5:22:02 PM it.unitn.disi.webarch.facchinetti.http.handler.GETRequestHandler handleProcessRequest
INFORMAZIONI: Process started: /Users/claudiofacchinetti/Documents/UniTN/WebArchs/Lab/TinyHttpd/process/reverse.java
set 26, 2021 5:22:03 PM it.unitn.disi.webarch.facchinetti.http.handler.GETRequestHandler handleProcessRequest
INFORMAZIONI: Output received: amor
```

Figure 2: Logs produced by TinyHTTPd for the request

1.4 Problems faced

In this section are reported what have been the issues faced during the development of this assignment. The only real issue faced have been the development of the code reported in the previous snippet, where the external process is started and the output is retrievered. In particular the problem was with the fact that in first place the `Runtime` API was used; this API does not allow to do many things in a clean way, like setting the directory or redirecting `stderr` to `stdout`. To overcome this the `ProcessBuilder` API was used instead.
Other than this there have not been many problems related to the assignment, but to the development of some of the features of the web server, like the templating language (see chapter 3 for that).

2 Part two: development with cgi-bin

2.1 Introduction

During the lectures it has been discussed of how `cgi-bin` have been one of the first mechanisms to allow developers to provide dynamic content; in particoular the Apache web server allows the execution of files located in a specific folder, named `cgi-bin` or `fcgi-bin`, in order to use their output as response to the HTTP request.
In the second part of the assignment we were asked to install the Apache web server, either directly or with some distribution like MAMP or XAMPP, and to write a bash/batch script that would have invoked the same Java program `reverse` with parameters and returned its output as HTTP response using the mechanism of `cgi-bin`.
For this specific case it has been installed the Apache web server by installing MAMP via the packet manager Homebrew using the following command

```
brew install --cask mamp
```

2.2 Solution

In order to perform the task it has been created a `.sh` file which does the following:

1. parse the query string in order to extract the parameters;
2. if there is no parameter named `par1` in the query string then show an error message;
3. otherwise it invokes the Java program by passing it the parameter.

In order to perform the first task it has been used one of the features introduced by Bash version 4, which is associative arrays. In particular the query string has been parsed and each parameter has been stored in a map, named `array` in the script, using its name as key.
Here it is the content of the `reverse.sh` script. Note that for semplicity the file `reverse.java` has

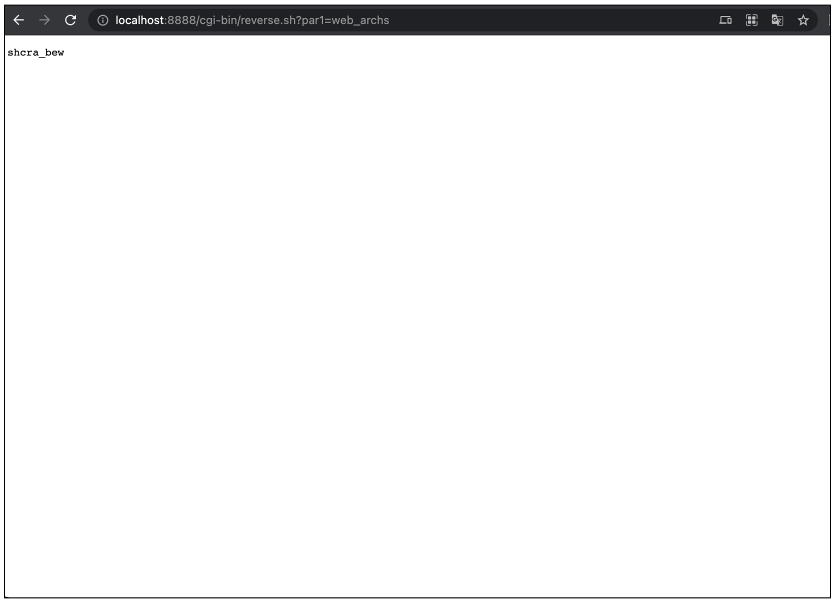
```
#!/bin/bash
saveIFS=$IFS
IFS='=&'
parm=($QUERY_STRING)
IFS=$saveIFS

declare -A array
for ((i=0; i<${#parm[@]}; i+=2))
do
    array[${parm[i]}]=${parm[i+1]}
done

echo "Content-type: text/plain; charset=iso-8859-1";
echo
if [ -z ${array['par1']+x} ];
    then echo "You must provide a parameter named 'par1'";
    else echo $(cd java && java reverse.java -par1=${array['par1']});
fi
```

Snippet 2: File `reverse.sh`

been copied inside a folder named `java` there the script is located.
This script has been placed in the `cgi-bin` folder where MAMP has been installed; in this specific case it has been placed in `/Applications/MAMP/cgi-bin/`.

2.3 Application running

Here it is a screenshot of the application running: the fact that it is now using the `cgi` mechanism is visible from the URL: now we are now querying the path `/cgi-bin/reverse.sh`.



```
shcra_bew
```

Figure 3: Result of calling the `cgi-bin` version with Chrome

3 Extra: introducing JTML (Java TeMplating Language)

As said at the beginning of the report the web server used for the first part of the assignment is not the one provided by the professor but a freshly created one. While creating this web server it has been explored the idea of including into this small project an extra feature: the possibility to define inside a static HTML page some particular code that would have been interpreted by the web server in order to produce dynamic content.
Even though it started as a simple way of embedding already existing Java variables inside an HTML page dynamically, it ended up being a small miniaturized version of a programming language which gets evaluated by an interpreter. Of course it has not all the features of a modern programming language, but gets its job done.
An example of using this is available in the `index.html` contained in the IntelliJ project delivered, enclosed inside the `jtml` tag. Please note that only the code enclosed in such tag gets evaluated.

3.1 Expression evaluation

The first thing that is doable with JTML is name evaluation: if we simply write an expression then the whole block will get evaluated to the `String` representation of that expression. Please note that at the current state only the last evaluation will substitute the block.
As an example the follwing code gets evaluated to ”Hello World”.

```
<jtml>
    "Hello World"
</jtml>
```

3.2 Variables

As every programming language JTML allows the developer to declare variables: they are declared with the straight forward notation `name:  type = value`. Here it is an example of how variables are declared.

```
<jtml>
    myString: java.lang.String = "Hello World"
    myInt: java.lang.Integer = 3
</jtml>
```

Please note that the declaration of a variable requires the developer to specify the `complete name` of the type it is willing to use.
It is also possible to make available to the page some default variables: this is doable by adding them to the NameResolver used by the interpreter before the actual interpretation happens.
It is also possible to create instances of other classes; theoretically speaking it is possible to create an instance of any Java class which is in the classpath of the TinyHTTPd application, however at this time it is only possible to instantiate classes which have a constructor accepting one single parameter. To instantiate one of such classes it is possible to assign the variable of that type to an instance of the class its constructor takes as only parameter, after the discussion on method calls.

3.3 Method invocation

As the base is Java it is also possible to invoke methods on variables either when we are performing an assignment or when we are willing to make the result of the expression substitute the block.
Method invocation is done by using the simple construct `variable -> methodName(param1, ..., paramN)` and it is also possible to perform a chain of method invocations such as `variable -> methodName(param1, ..., paramN) -> method1() -> method2(par1)`.
Please note that at the time of writing any parameter passed to the method can be either a variable or a literal, keeping its type or using it as a constructor parameter for another class as discussed before. It is not possible at the moment to provide as a parameter an expression involving another method call.

3.4 Final example

Here there is an example using all the concepts, but it is also possible to find a different in the `index.html` file delivered in the IntelliJ project.

```
<jtml>
    myString: java.lang.String = "Hello world" -> toUpperCase()
    length: java.lang.Integer = myString -> length()
    myStringBuffer: java.lang.StringBuffer = myString -> substring(3,length)

    myString: java.lang.String = myStringBuffer -> reverse()
    myString -> toLowerCase()
</jtml>
```

3.5 Final remark

As a final remark I would like to remember that JTML has been developed only for the sake of experimenting and having fun while studying and not to become the next PHP; as so it is full of bugs, security issues and it is also missing many features. Please consider it as an experiment and not as a full fledged programming language, even if it would be interesting to develop it even more.