Apache Tomcat as a transport in CXF

Final report

Team and contact information:

Ivan Kravchenko (ivan.kravchenko@students.unibe.ch)

Guillaume Corsini (guillaume.corsini@students.unibe.ch)

13. June 2019

Version 3.1

Table of contents

Version monitoring

Monitoring of the version in this project organization document enables following modification activities progress within the project. The version is changed only if a major change takes place. Otherwise, only a decimal is changed (1.0 to 1.1)

Version	Update date	Chapters modified
1.0	9. June 2019	initial draft
2.0	10. June 2019	Results: CXF
3.0	12. June 2019	Results: proof of concept Recommendations
3.1	13. June 2019	Added links, fixing typos and other small issues.

Executive summary

The goal of this project was to deliver a proof of concept where an embedded Tomcat web server is used as a transport in the Apache CXF framework. In order to achieve this goal, the required classes were implemented into the CXF framework. Please note, that these classes still contain certain limitations that should be fixed in the next iteration. The proof of concept contains three examples each using a different web service (JAX-WS, REST or SOAP).

1. Project description

CXF is a Java open-source framework, which simplifies building web services. Tomcat is a popular web server written in Java. The project intends to integrate Tomcat in CXF as an embedded server with a purpose of transporting HTTP requests. In the following chapters, the use of the words “embedded Tomcat” will be shortened by writing only “Tomcat”.

1.1. Project context

Red Hat is a company providing many services for enterprises mainly based on open source software. Being promoters of open source code, they publish all of their developed code. One of the many projects that Red Hat is actively developing is the Tomcat server. The tomcat server is an open source web server for Java programming language. In order for it to continue being popular in the future, Red Hat wants to have it integrated into the Apache CXF framework. Our project should enable the connection of these two tools.

1.2. Goals and objectives of the project

The major goal is to deliver a proof of concept that Tomcat can be used as a transport in CXF similarly to Jetty or Undertow. This will be achieved by reaching the following objectives:

1. Study and understand the following technologies: CXF (especially transport requirements), how embedded Jetty and Undertow servers are used as transports in CXF.
2. Study how an embedded Tomcat can be added to the CXF framework.
3. Propose a concept resulting in an embedded Tomcat being used as a transport in CXF.
4. Implement the aforementioned concept and write a multitude of tests.
5. Provide a proof of concept by developing a simple application which uses an embedded Tomcat as a transport in CXF.

The implementation contains a lot of documentation ensuring that others could take over the project in order to finish all the implementations or to make further adaptations.

2. Methodology

The programming language is Java, explicitly the versions 8 and 11. Both students used IntelliJ from JetBrains as their IDE environment.
All of the code will be published under an ASF License.
The team used GitHub for code synchronization. The initial code was also forked from the official GitHub repositories.

3. Results and realization

The major goal of creating a proof of concept using an embedded Tomcat as a transport in CXF succeeded. This has been achieved by reaching each objective set at the beginning of the project: The concept of how Tomcat could be used a transport is based on similar implementations for Jetty and Undertow. Then, the key components of the concept were implemented one after another. The correct functionality was checked by writing unit tests and several integration tests. Finally, the proof of concept was created in order to demonstrate that an embedded Tomcat server can now be used as a transport in CXF.

3.1. CXF

The codebase of Apache Tomcat being embedded as a transport in CXF can be found here:

https://github.com/gcorsini/cxf/tree/v1.0-rc

This is a fork of the official CXF project from GitHub. It contains the branch *transport_tomcat*. In order to build this project, it is necessary to run *mvn clean install*. Please note that at the time of this writing, several flags were required due to issues with other modules. Hence, the command was extended with three flags to skip running module tests:

mvn clean install -DskipTests -Dcheckstyle.skip -Dpmd.skip=true

If the IDEA used complains about missing classes, it is very probable that some generated folders were not marked as source code. It is possible to either add each generated folder manually or adapt the IDEA settings.

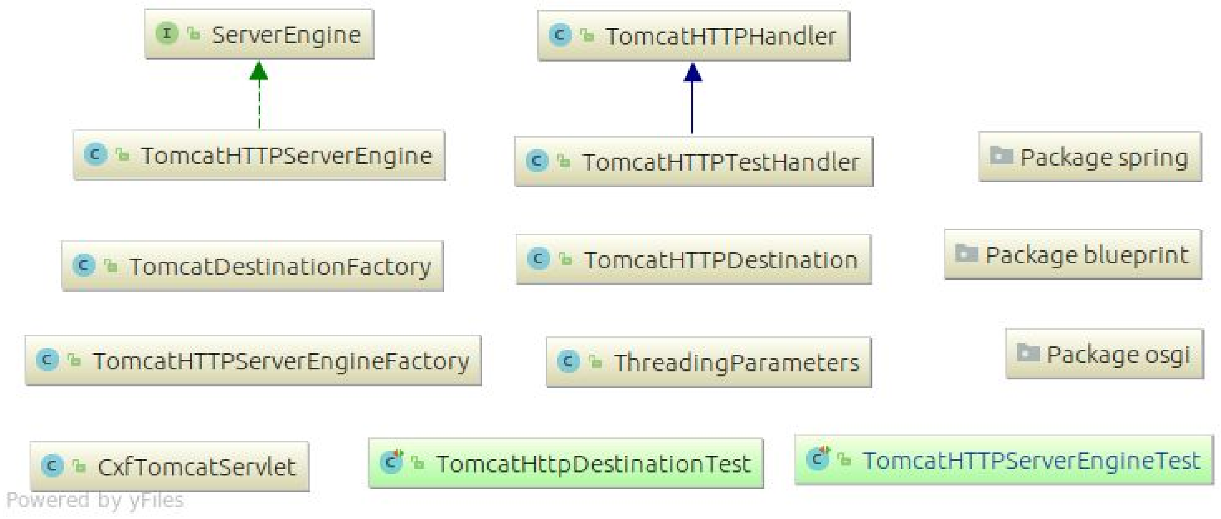
The implementation of Tomcat as a transport is found at the location:

rt/transports/http-tomcat/

The unit tests are also located there, but one of the integration tests is located here:

systests/transport-tomcat/

The class structure of the implementation is shown in the diagram below:



Three important classes are *TomcatHTTPServerEngine*, *TomcatHTTPHandler* and *TomcatHTTPDestination*:

Class *TomcatHTTPServerEngine* starts and manages the embedded Tomcat server lifecycle. It also adds and removes servlets.

Class *TomcatHTTPHandler* implements a Tomcat Filter and ensures that requests and responses are handled correctly.

Class *TomcatHTTPDestination* binds the client with the Tomcat server instance.

3.2. Proof of concept

The proof of concept using three different web services (JAX-WS, REST and SOAP) can be found here:

https://github.com/gcorsini/quick_rest_tomcat/tree/v1.0

While the Readme file on that project contains a very short summary of the required steps, here is a detailed guide on how to run the demonstrations:

1. Ensure that the embedded tomcat server has been merged into the original CXF codebase.
    a. If this is the case, continue with step 2.
    b. If this is **not** the case:
        i. Pull the code from the link specified in Chapter 3.1.
        ii. Checkout the branch *transport_tomcat*.
        iii. Run *mvn clean install*. If this produces issues or errors, you should append the following flags creating the complete command:
        mvn clean install -DskipTests -Dcheckstyle.skip -Dpmd.skip=true
        iv. Return to the repository containing the proof of concept.
2. In the pom.xml of each web service comment the jetty dependency and uncomment the tomcat dependency.
3. Run *mvn clean install* (in the proof of concept repository **without any flags**).
4. Start the server of one web service, then start its corresponding client and check the output. Repeat for every other web service. (Also, remember that step 2. might be required to use an embedded Tomcat instead of Jetty.)

4. Recommendations

4.1. Statement of recommendations

The following recommendations are made to the best of our knowledge. They are mostly based on the comparison between Jetty, Undertow and Tomcat as a transport in CXF. The second limitation regarding https is due to complications that appeared during development. Now, that the required settings have been identified, it is fairly simple to fix them. On the other hand using a variable in order to get the security parameters, enables a lot more options that should all be taken into consideration.

4.2. Limitations

Currently, the biggest limitation is the fact that only one servlet can be served by an embedded Tomcat at a time. In order to enable support for multiple servlets on the same server, the method *addServant* has to be extended

While using the https protocol for a servlet could theoretically be used, it is limited to the localhost and the specifications of the test class. Therefore, *TLSServerParameters tlsServerParameters* in the *TomcatHTTPServerEngine* should be used to set up a https connection. The parameters should replace current default values that are set in the method addServant after the line containing the comment *Use HTTPS*.

Analog to the previous parameter for the TLS server settings, the current implementation is ignoring any multithreading parameters. In order to implement these, knowledge of multithreading in Tomcat is desired. Then, the *ThreadingParameters threadingParameters* in the *TomcatHTTPServerEngine* can be used when creating the server in the method *addServant*.

It seems like Jetty and Undertow accept to set multiple handlers for a server. While this can be achieved by daisy-chaining multiple filters by overriding the *TomcatHTTPHandler*, the list of filters called *handlers* in the *TomcatHTTPServerEngine* should be taken into consideration when adding a servlet with the method *addServant*.

Additionally, some less significant options can not be set in the current state. One of these options is setting the maximum idle time of the embedded server.

4.3. Outstanding issues and perspectives for future work

First, the limitations mentioned above should be implemented. This would complete the requirements for Tomcat as a transport in CXF.

Second, adding more unit tests in order to cover the added functionalities is highly recommended.

Third, creating more integration tests in order to check the correct behaving for all different possibilities for Tomcat as a transport in CXF. It would probably be possible to implement many from the folder distribution/src/main/release/samples.

Finally, on GitHub: The latests commits from the original fork have to be pulled and merged into the current codebase. Afterwards, all commits of the branch *transport_tomcat* should be squashed and a merge request sent.