New Relic Java RMI Integration Sample User Guide

Build and Deploy: RMI Sample Client, Sample Server, RMI Extension & Integration with new relic java agent

What is RMI ?

RMI stands for Remote Method Invocation. It is a Java-based application programming interface (API) that allows objects in one Java virtual machine (JVM) to invoke methods on objects located in another JVM, whether they are on the same machine or on a remote system. RMI enables distributed computing by providing a mechanism for objects to interact and communicate with each other across different JVMs.

With RMI, a client program can access and invoke methods on remote objects as if they were local objects, abstracting the complexities of network communication and serialization. RMI handles the low-level details of communication, such as marshaling and unmarshaling parameters and results, so developers can focus on writing the application logic.
RMI relies on Java's object serialization mechanism to pass objects between JVMs. It allows objects to be passed as parameters, return values, or exceptions in method invocations, enabling distributed computing scenarios such as remote method calls, distributed object models, and distributed event notification systems.

RMI has been a popular choice for building distributed Java applications, although alternative technologies like Java Message Service (JMS), Enterprise JavaBeans (EJB), and more recently, web services and RESTful APIs have also gained prominence.

The Sample Code:

The provided code consists of several Java classes that work together to implement a simple RMI-based character counting application. Let's go through each class and understand its role within the application:

CharacterCounter.java:

- This interface extends the Remote interface, indicating that it's designed for remote communication.
- It declares a single method, countCharacters, which takes a String parameter and returns an int.
- The method throws a RemoteException, indicating potential issues with remote communication.

CharacterCounterImpl.java:

- This class implements the CharacterCounter interface and extends UnicastRemoteObject, which facilitates remote object functionality.
- It overrides the countCharacters method to provide the implementation for counting the characters in the input text.
- The implementation internally uses a FactorMultiplier object to multiply the character count by 100.
- It extends UnicastRemoteObject to enable the object to be used in RMI communication.
-

FactorMultiplier.java:

- This class provides a helper method called MultiplierByHundred that takes an integer and multiplies it by 100.
- It is utilized by the CharacterCounterImpl class to multiply the character count by 100 before returning it.

RMIServer.java:

- This class acts as the server-side entry point for the RMI application.
- It instantiates a CharacterCounterImpl object and binds it to the RMI registry with the name "CharacterCounter".
- The RMI registry allows clients to locate and communicate with the registered remote objects.
- Upon successful binding, it prints a message indicating that the server is ready.

CharacterCountServlet.java:

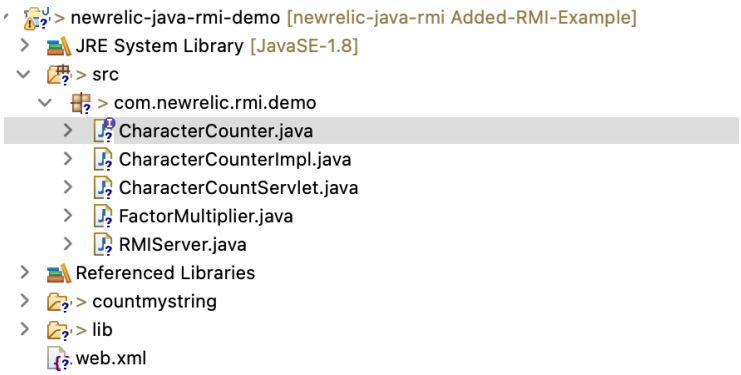
- This class is a servlet that handles HTTP POST requests at the /character-count URL.
- It receives a text parameter from the request and attempts to count the characters remotely.
- It uses the Naming.lookup method to locate the remote CharacterCounter object by its RMI URL.
- Upon successful lookup, it calls the countCharacters method on the remote object to get the character count.
- It sends the character count as the response to the HTTP request.

Overall, this code demonstrates a simple client-server architecture utilizing RMI for remote communication. The server-side code (RMIServer, CharacterCounterImpl) exposes the CharacterCounter interface as a remote object, while the client-side code (CharacterCountServlet) communicates with the server to count the characters in a given text.

Building and deploying the application with New Relic Java Agent

1- Import the Sample Code project in eclipse editor and build the code

```
newrelic-java-rmi
└── example
    └── newrelic-java-rmi-demo
```



2- Verify in the bin folder you will have following classes available

ls com/newrelic/rmi/demo/

CharacterCountServlet.class CharacterCounter.class CharacterCounterImpl.class FactorMultiplier.class RMIServer.class

3- Create a war file and deploy on tomcat server

Go to newrelic-java-rmi/example/newrelic-java-rmi-demo/countmystring

Update your class files in following demo location

```
├── META-INF
│   ├── MANIFEST.MF
│   └── war-tracker
├── WEB-INF
│   ├── classes
│   │   └── com
│   │       └── newrelic
│   │           └── rmi
│   │               └── demo
│   │                   ├── CharacterCountServlet.class
│   │                   └── CharacterCounter.class
│   └── web.xml
```

```
Java
jar -cvf countmystring.war *
```

Deploy this war file on tomcat server running with JDK 1.8 [Preferred]

4- Configure NewRelic Java Agent for Tomcat Server by setting the following and ReStart the Tomcat Server.

```
Java
export CATALINA_OPTS=' -Dnewrelic.config.app_name=Tomcat_RMI_Client
-javaagent:<absolutePathOfNewRelicAgentInstallation>/newrelic.jar'
```

5- run *rmiregsitry* from ***newrelic-java-rmi-demo/bin*** folder where interface class is available in current path and keep it running

```
Java
rmiregistry
```

6- run RMI Server in one of the command prompt from ***newrelic-java-rmi-demo*** folder and keep it running and you will be seeing Message “RMIServer ready”

```
Java
java -javaagent:/Users/gsidhwani/gulab-nr-java-agent/newrelic/newrelic.jar
-Dnewrelic.config.app_name=RMI-SERVER -cp "./bin"
com.newrelic.rmi.demo.RMIServer
```

So, now both standalone RMI Server and Web Servlet have deployed and are ready for the test.

7- Run the following command to make a post request using *curl* command

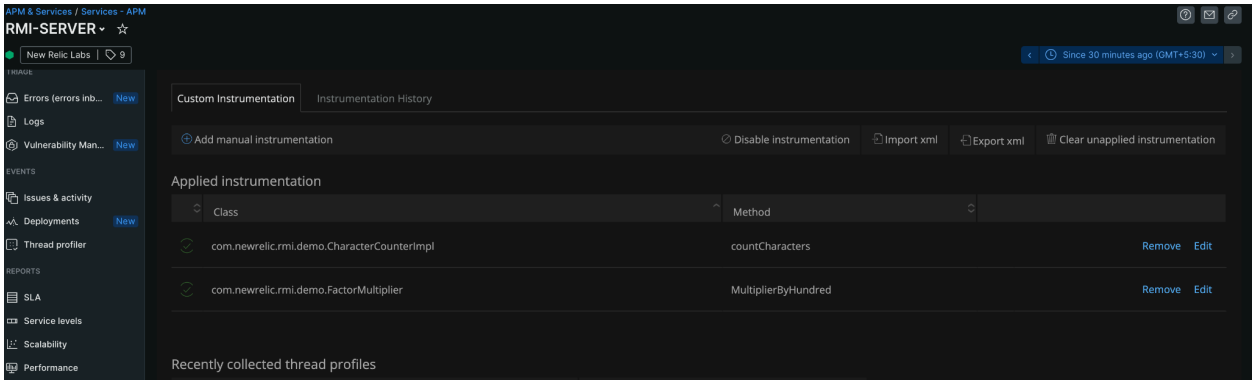
```
Java
curl -X POST -d 'text=Hello World'
http://localhost:8080/countmystring/character-count
```

```
Character count: 1100
```

It will return the number of characters in provided text multiplied by 100.

Analyzing the applications at New Relic One Platform without RMI Extension

As the New Relic Agent is already configured and hence it will instrument the Sample Servlet by default but our sample RMI Server will not be instrumented by default So, we will add classes and methods to be instrumented at New Relic Instrumentation Editor



You may also import this XML to instrument our Sample Class and Method.

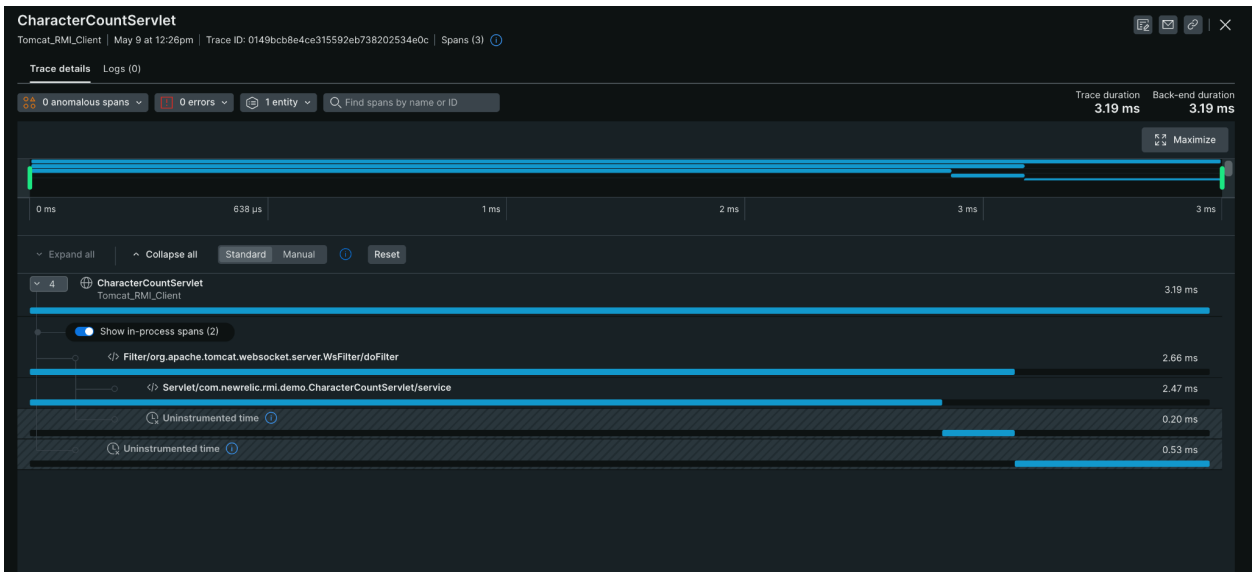
Java

```
<?xml version="1.0" encoding="UTF-8"?>
<extension xmlns="https://newrelic.com/docs/java/xsd/v1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="newrelic-extension
extension.xsd" name="extension-example" version="1.0" enabled="true">
  <instrumentation>
    <pointcut transactionStartPoint="false" ignoreTransaction="false">
      <nameTransaction/>
      <className>com.newrelic.rmi.demo.FactorMultiplier</className>
      <method>
        <name>MultiplierByHundred</name>
      </method>
    </pointcut>
    <pointcut transactionStartPoint="true" ignoreTransaction="false">
      <nameTransaction/>
```

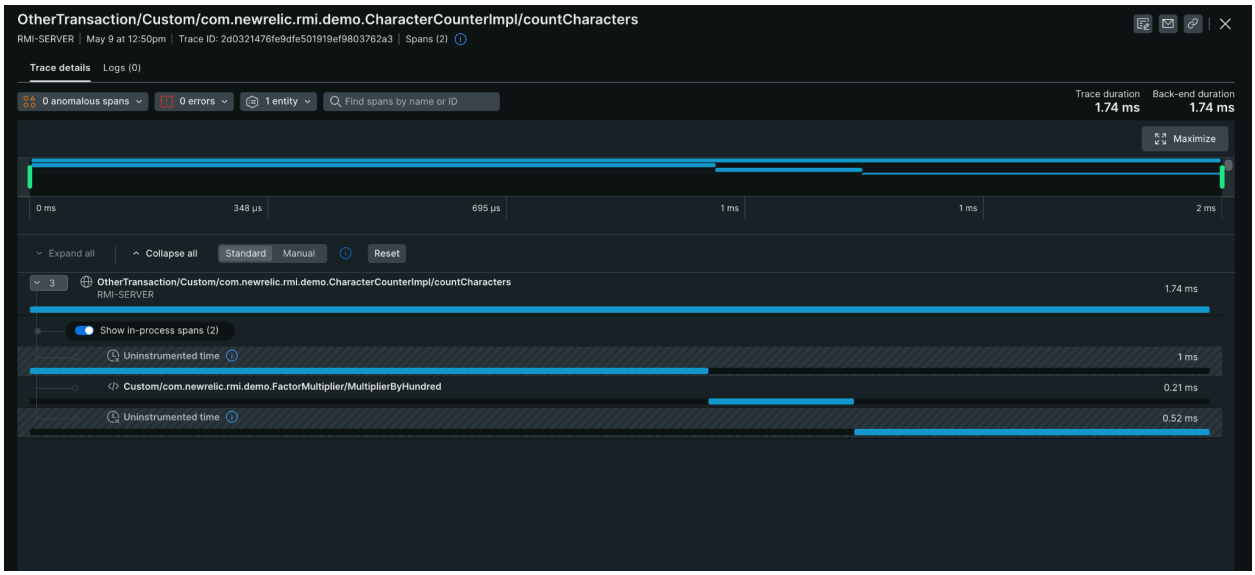
```
      <className>com.newrelic.rmi.demo.CharacterCounterImpl</className>
      <method>
        <name>countCharacters</name>
      </method>
    </pointcut>
  </instrumentation>
</extension>
```

Please apply these changes and run the Transactions using Step 7 .

Now, Open Distributed Tracing of **Tomcat Application** and you may see that only Servlet Service is reported and there is no call being reported for RMI Server.



Open Distributed Tracing of **RMi SERVER** and you may see that your instrumented methods are being visible but there is no caller Entity information available to analyze this further.



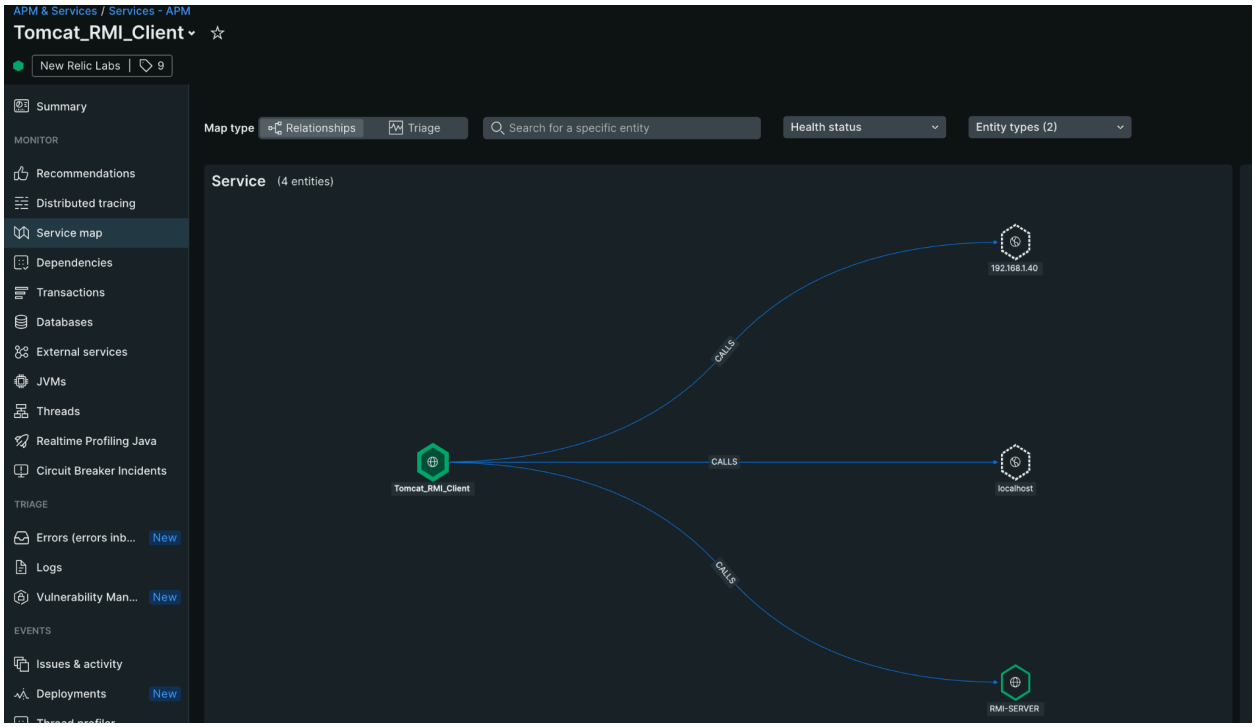
Both applications are reporting their respective traces but these are not interconnected in Distributed traces.

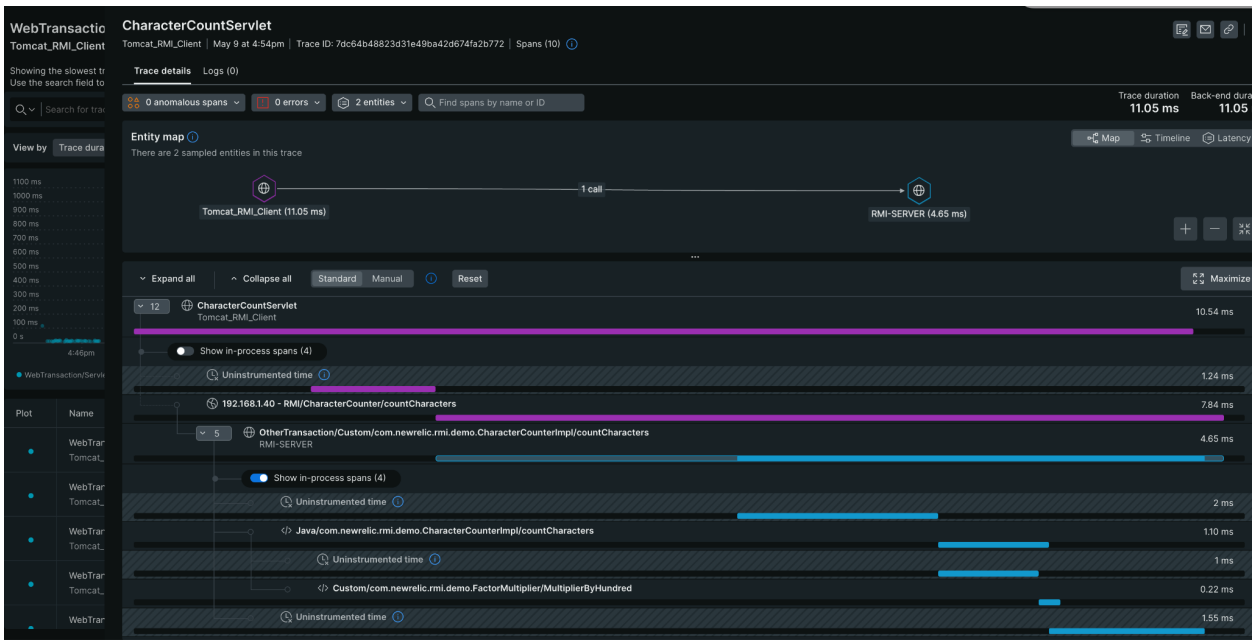
Analyzing the applications at New Relic One Platform with the RMI Extension

Please download the extension jars [rmi.jar and rmi-stubs.jar] and put these to the newrelic/extensions folder and run the transaction again.

.
newrelic
├── extensions
│   ├── rmi-stubs.jar
│   └── rmi.jar

Upon analyzing the Distribution Traces of applications on the New Relic One platform, it becomes evident that the RMI client and RMI Server entities are now interconnected. This can be observed by referring to the provided snapshot from the Service Map and Distributed trace.





The rmi extension is a very useful plugin to analyze RMI based transactions running in distributed systems .

Need Help?

Get in touch using the email Gulab Sidhwani