Axon Immersion Training
Learn how to build scalable, extensible and high performance applications with a CQRS-based architecture.
MORE INFO

8

Digg

HOW TO REMOTELY DEBUG APPLICATION RUNNING ON TOMCAT FROM WITHIN INTELLIJ IDEA

July 14th, 2014 by Dadepo Aderemi

This post would look into how to tackle and debug issues in scenarios where they only occur in production (or other remote environment) but not in development environment. As anybody who has been in this kind of situation would acknowledge, trying to pinpoint the cause of these kind of "issues" might quickly end up being a practice at taking shots in the dark: a very time-consuming and

TRIFORK OFFICES

Aarhus
Amsterdam
Berlin
Copenhagen
Eindhoven
London
San Francisco
Zurich

ERLANG SOLUTIONS OFFICES

inefficient process.

It was this kind of situation I recently found myself, where, I had to rectify certain issues that were occurring in the production environment but could not be reproduced on the development machine.

Fortunately enough, the said issues could be reproduced in the testing environments (which is as close to the production environment as possible). But having the issues reproducible in the test environment was good In that it confirms the issues needed to be fixed, but it was of little help in actually tracking the issues down, finding the cause and fixing it. Relying just on log outputs was not enough…What if I could debug the test environment from my machine?

It was at this stage that I thought about remote debugging and if there exist the possibility of having an application run in a remote JVM and still be able to intercept its execution from the copy of the source code running on a local IDE: sounds just like what would get the job done.

And sure this is very possible. I looked into what is needed; the set up and all. It did not sound complicated. So, together with the help of a colleague, Thijs Schnitger, was able to get it up and running without much hassles.

This post thus describes the procedure of setting up the ability to remotely debug a JVM application from within an IDE. The post outlines the procedure using IntelliJ IDEA as the IDE, and the remote application to be debugged would be web application running on Tomcat. The steps outlined below should apply with any remote JVM application and any IDE, although the exact steps may differ, the general idea would remain same. The post also gives a brief overview of the technologies that makes remote debugging possible.

The Configuration Instructions

The process of getting remote debugging working involves two steps.

1. Starting Tomcat with remote debugging enabled and
2. having your IDE, in our case IntelliJ IDEA, to be able to debug the remote tomcat application.

There are couple of ways to get the first part done and it slightly differs depending on which OS environment your Tomcat instance is running on. But, regardless of the method used, the main idea behind the configuration remains the same; which is: pass specific start up options to the JVM that would enable remote debugging.

The JVM start up arguments needed to have remote debugging activated can be set via

Trifork is recruiting new talent now!
Check out our career page.

JPDA_OPTS, *CATALINA_OPTS* and *JAVA_OPTS* although using *JAVA_OPTS* is not usually advised, reason being that the setting specified via *JAVA_OPTS*, is exposed to all JVM applications, but with *CATALINA_OPTS* the settings defined is limited only to Tomcat.

USING JPDA_OPTS

Using the *JPDA_OPTS*, option you would have the needed start-up argument set in a file named setenv.sh (or setenv.bat if on windows). Create the file if it does not exist already. Have it in the *CATALINA_HOME/bin* directory. And add this to the content:

```
export JPDA_OPTS="-agentlib:jdwp=transport=dt_socket, address=1043,
server=y, suspend=n"
```

and if on Windows:

```
set JPDA_OPTS="-agentlib:jdwp=transport=dt_socket, address=1043,
server=y, suspend=n"
```

What these settings basically do is to enable remote debugging and configure available options: specifying the communication protocol between the running application and the debugger, (ie *transport=dt_socket*) the port at which the remote application should debugged (*ie address=1403*). The *server=y* setting indicates that this JVM would be the one to be debugged while *suspend=n* is used to tell the JVM to execute right away and not wait for an attached debugger. If set to "y" then the application would be suspended and not run until a debugger is attached.

A good situation where you would want to have *suspend=y* is when debugging an issue that prevents an application from starting successfully, having *suspend=y* would make sure that the JVM waits for the remote debugger to connect before attempting to start and run the application.

Although the settings can be put directly inside *catalina.sh* (or *catalina.bat*) it is always preferable to have extra configurations in the *setenv.** file. It would be automatically be picked up by Tomcat.

Note that another options you may encounter that may be used to enable remote debugging is:

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=1043, suspend=n
```

The difference between this and the recommended setting is that the *-Xdebug* and *-Xrunjdwp* option is the old way of enabling remote debugging; it applies to JVM prior to JAVA 5.0 (JAVA 1.5.0) while -agentlib:jdwp option applies to JAVA 5.0 and beyond…And with this configuration in place, you can then start Tomcat via the command line arguments:

GOTO Chicago 2016
Conference: May
Workshop: May

GOTO Amsterdam 2016
Workshop: June 13
Conference: June 14-15

GOTO London 2016
Workshop: October 12
Conference: October 13-14

GOTO Copenhagen 2016
Conference: Oct. 3-4
Workshop: Oct. 5-6

CATEGORIES

- Business (86)
  - Agile (7)
  - Education (16)
  - Recruitment (6)
  - System Administration (13)
- Content Management (19)
  - Hippo (10)
  - Liferay (3)
  - Magnolia (4)
- Custom Development (293)
  - AngularJS (3)
  - C# (3)
  - DevOps (7)
  - From The Trenches (15)
  - Frontend (16)
  - Groovy (6)
  - GWT (6)
  - Java (76)
  - Objective C (4)
  - QTI (1)
- Docker (1)
- General (3)
- Knowledge (89)
  - Conference (70)

```
$CATALINA_HOME/bin/catalina.sh jpda start
```

USING JAVA_OPTS OR CATALINA_OPTS

If you have Tomcat running as a windows service, then configuring Tomcat to start up with ability to be debugged remotely is done by simply specifying the start up arguments in the run properties.

Open up the Apache Tomcat properties dialog box, and under the Java tab add the required start up option:

```
-agentlib:jdwp=transport=dt_socket,address=1043,server=y,suspend=n
```

Make sure that each entry is on a new line and there are no spaces between the options

With this added to the options, starting the Tomcat service would have remote debugging enabled.

If per chance you are not running Tomcat on Windows as a service, to enable remote debugging, you need to specify the required options in the setenv.bat file:

```
set CATALINA_OPTS="-agentlib:jdwp=transport=dt_socket,address=1043,server=y,suspend=n"
```

If you are running on linux you have:

```
export CATALINA_OPTS="-agentlib:jdwp=transport=dt_socket,address=1043,server=y,suspend=n"
```

Start Tomcat like you would normally then do by running the `catalina.bat` or `catalina.sh` script.

STARTING TOMCAT WITH JPDA

Another way of running Tomcat with remote debugging is to use the JPDA switch, to start Tomcat; this would automactically have remote debugging enabled with its options set to default values.

For example, this:

```
catalina jpda start
```

would start Tomcat with the following settings:

```
-agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=n
```

- Newsletter (19)
  - Trifork NewsBites 2014 (1)
- Training (15)
- Mobile (39)
  - Android (10)
  - Apple Watch (1)
  - HTML5 (4)
  - iPad (11)
  - iPhone (14)
  - Windows Phone (2)
- NoSQL & Search (109)
  - Apache Mahout (13)
  - Enterprise Search (97)
    - Apache Lucene (30)
    - Apache Solr (17)
    - Elasticsearch (33)
  - NoSQL (7)
    - MongoDB (5)
    - Riak (1)
- Spring (11)
- Swift (1)

TAGS

Agile Amsterdam Android Angularjs Apache Apache Lucene Apache Mahout Apache Solr Architecture Axon Framework Best Practices CMS

But what if you want to change any of these default settings and still use the JPDA start switch? This can be done by setting the appropiate environment variable for the options as required by Tomcat. The environment variables are:

JPDA_TRANSPORT: to specify JPDA transport used
JPDA_ADDRESS: to specify port for remote debugging
JPDA_SUSPEND: to specify if to suspend JVM after startup

so running:

```
export JPDA_ADDRESS="8080"
```

and then

```
catalina jpda start
```

would have remote debugging possible on port 8080.

Configuring IntelliJ IDEA

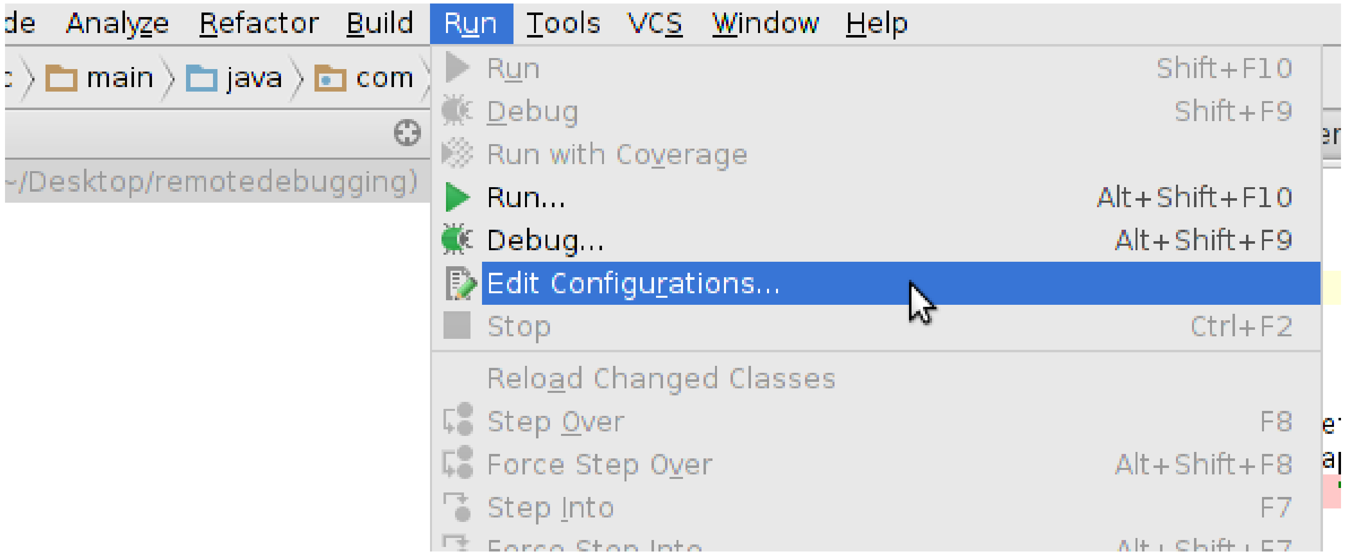
With the remote JVM running the Tomcat started with the required start up arguments, the next thing to do is to configure the debugger in IntelliJ IDEA.

There are two ways to this configuration: *The Remote Tomcat* settings options or *Remote settings* option.

Using The Remote Tomcat Settings Option.

Click on Run   Edit Configuration:

Conference
CQRS Customer DDD
Docker
Domain Driven Design
Elasticsearch
Empow erment
Enterprise Search
GOTO Amsterdam
GOTO Night Groovy GWT
Hadoop Hippo HTML
IOS IPhone Java
Javascript Linux
Objective C
Open Source
Performance Productivity
Relationship Search
Spatial Search



The *Edit Configuration…* settings dialog box, pops up.

Click on the **+** button on the top left   Tomcat Server   Remote. As shown below:

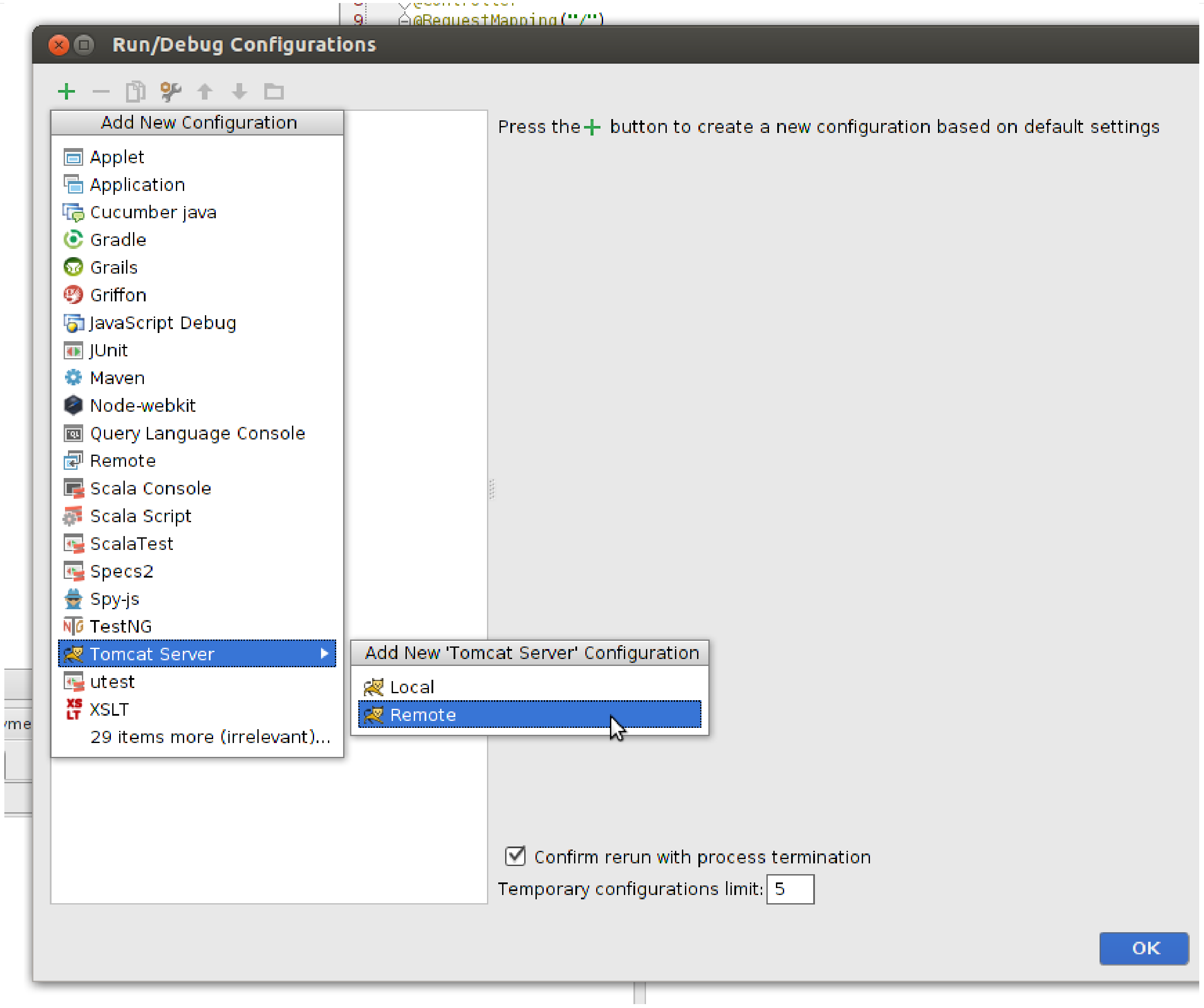
Spring
Spring Framework
Testing Training
Wicket

LATEST TWEETS

My Tweets

META

Log in
Entries RSS
Comments RSS
WordPress.org



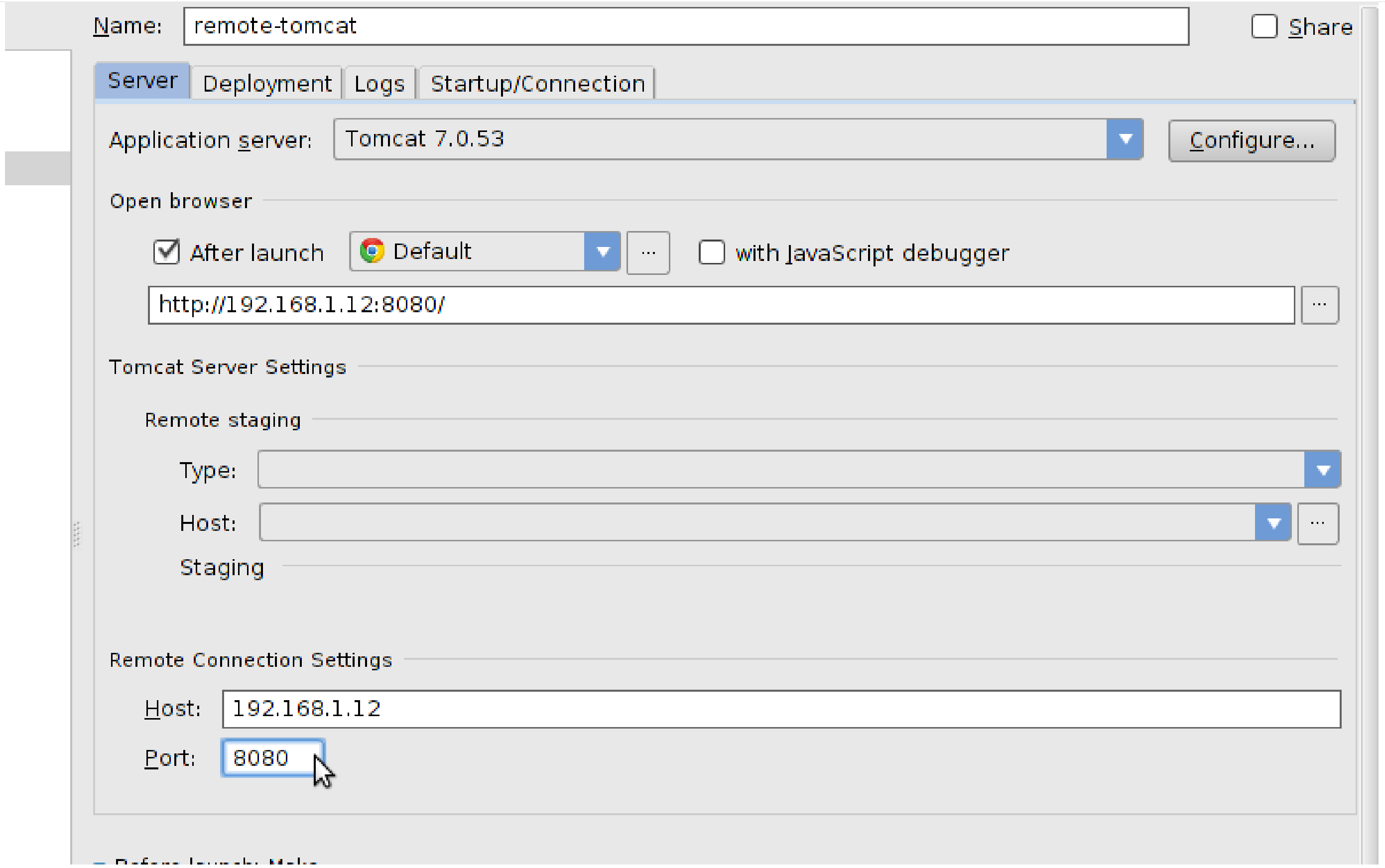
Press the + button to create a new configuration based on default settings

☑ Confirm rerun with process termination

Temporary configurations limit: 5

Next is to fill in the required settings.

On the **Server** tab, specify the *host details* and the *port* the *remote* Tomcat is running on:


Name: remote-tomcat

☐ Share

Application server: Tomcat 7.0.53

Open browser

☑ After launch Default ☐ with JavaScript debugger

http://192.168.1.12:8080/

Tomcat Server Settings

Remote staging

Type:

Host:

Staging

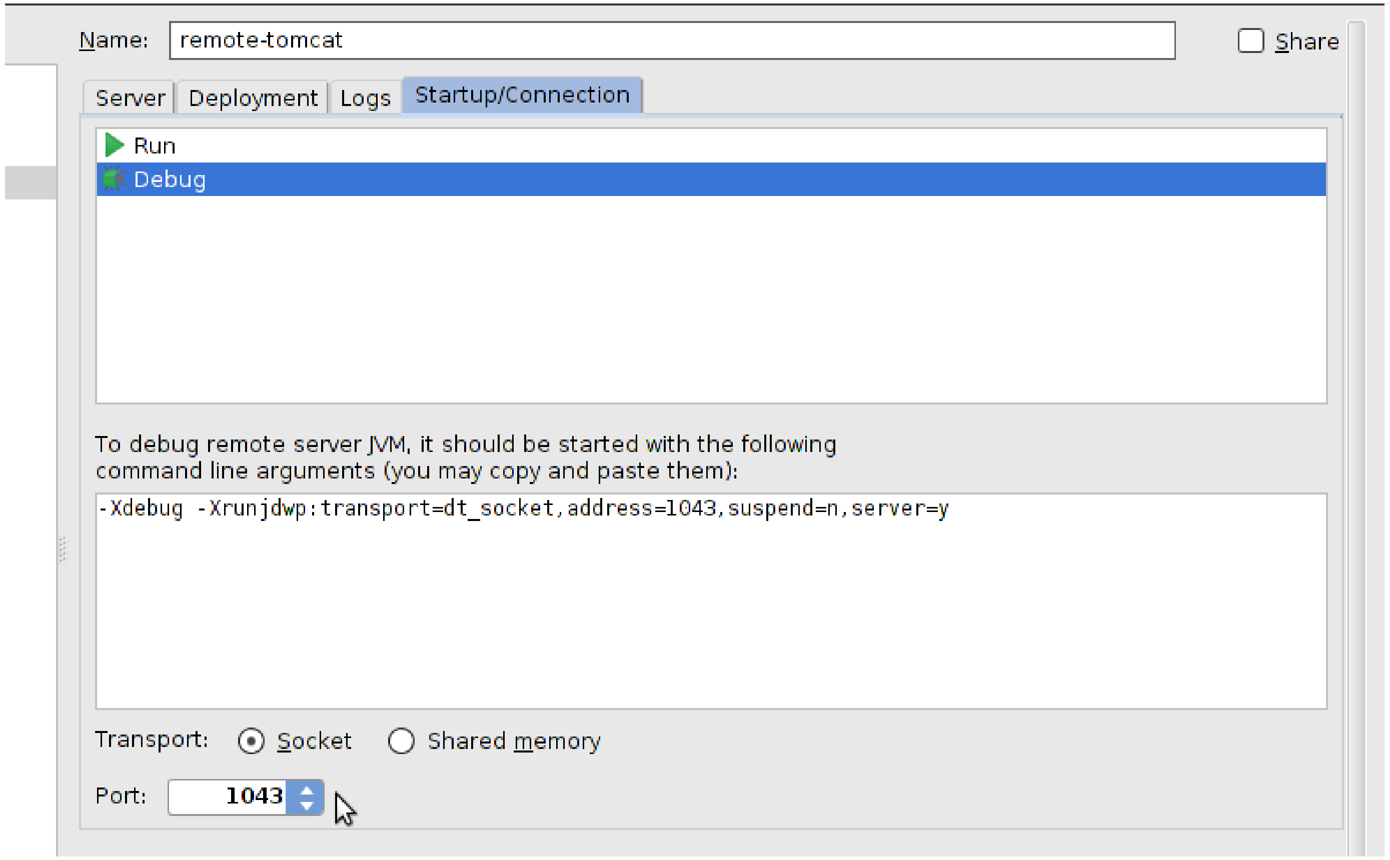
Remote Connection Settings

Host: 192.168.1.12

Port: 8080

Then switch to the **Startup/Connection** tab, where you would specify the details of the remote JVM needed to be debugged from intellij IDEA.

Click on **Debug** and specify the port you specified while configuring the Tomcat; *1043* in our case.


Name: remote-tomcat

Share

To debug remote server JVM, it should be started with the following command line arguments (you may copy and paste them):

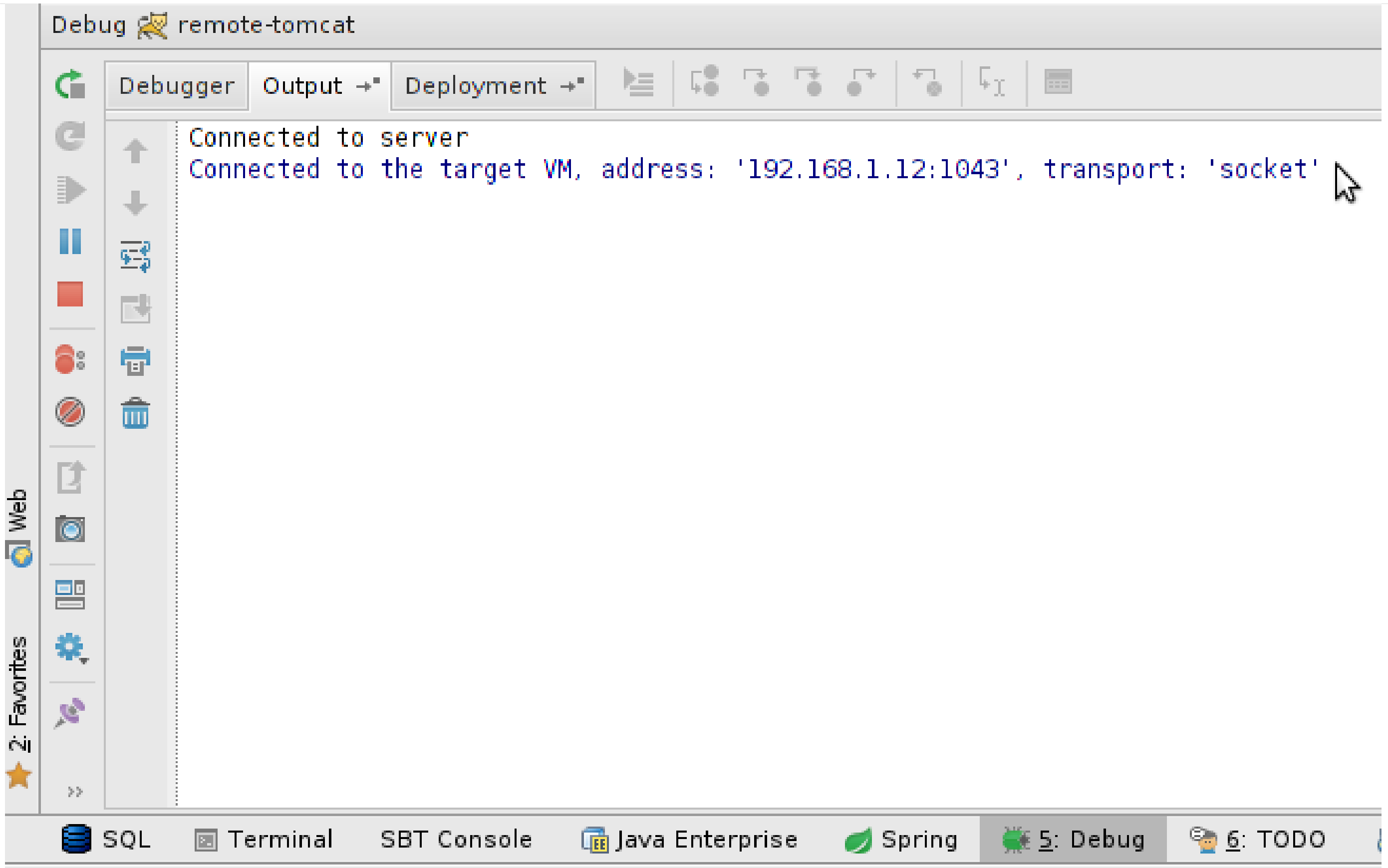
```
-Xdebug -Xrunjdwp:transport=dt_socket,address=1043,suspend=n,server=y
```

Transport: [✓ Socket] Shared memory

Port: 1043

And that is it. Click on **Ok** to save your changes, then start the debugging session by pressing the debug icon or by using the keyboard shortcut *(shift + f9)*

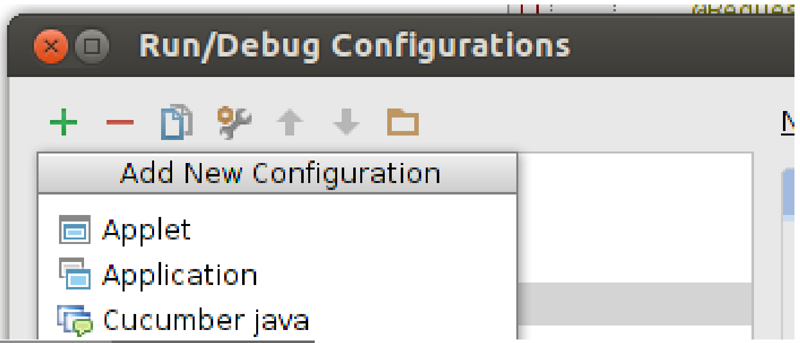
If all the configuration is set up correctly, you should see a notice that the IDE has successfully connected to the target VM in the Debug tab in the bottom console:

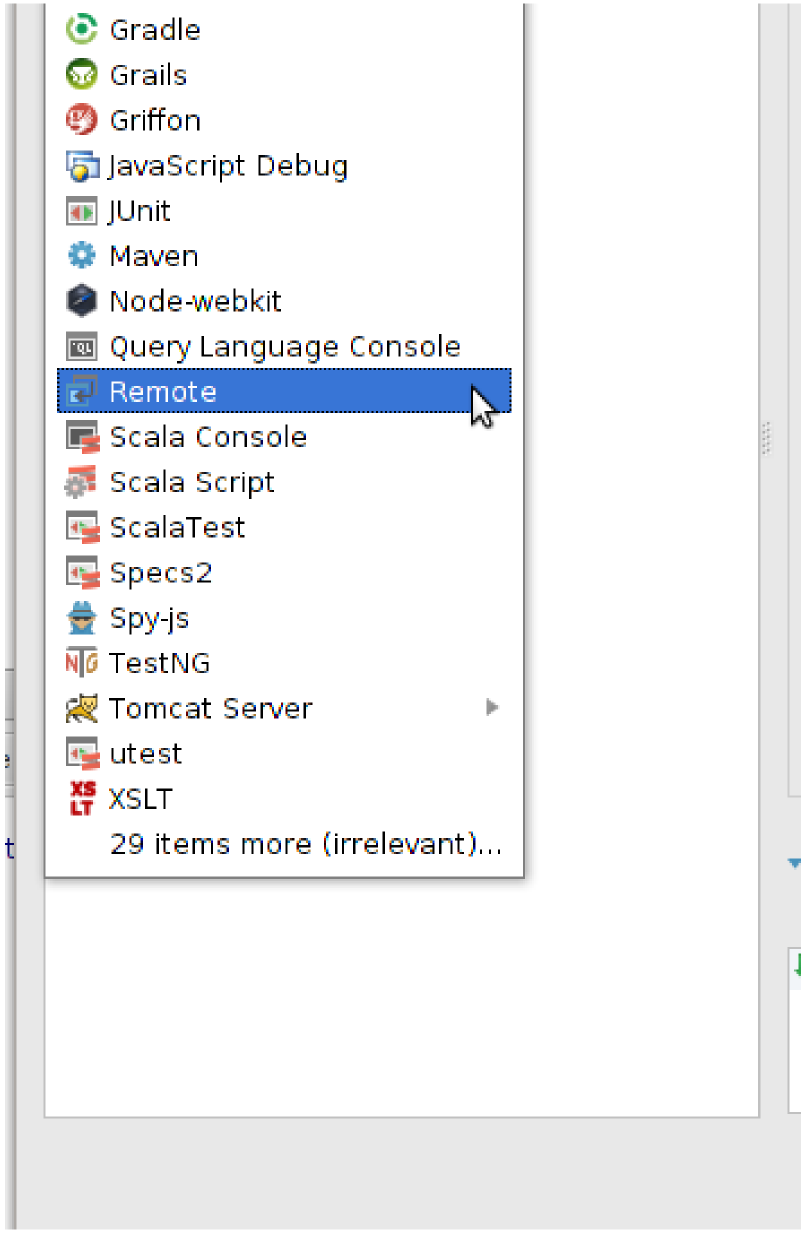


```
Connected to server
Connected to the target VM, address: '192.168.1.12:1043', transport: 'socket'
```

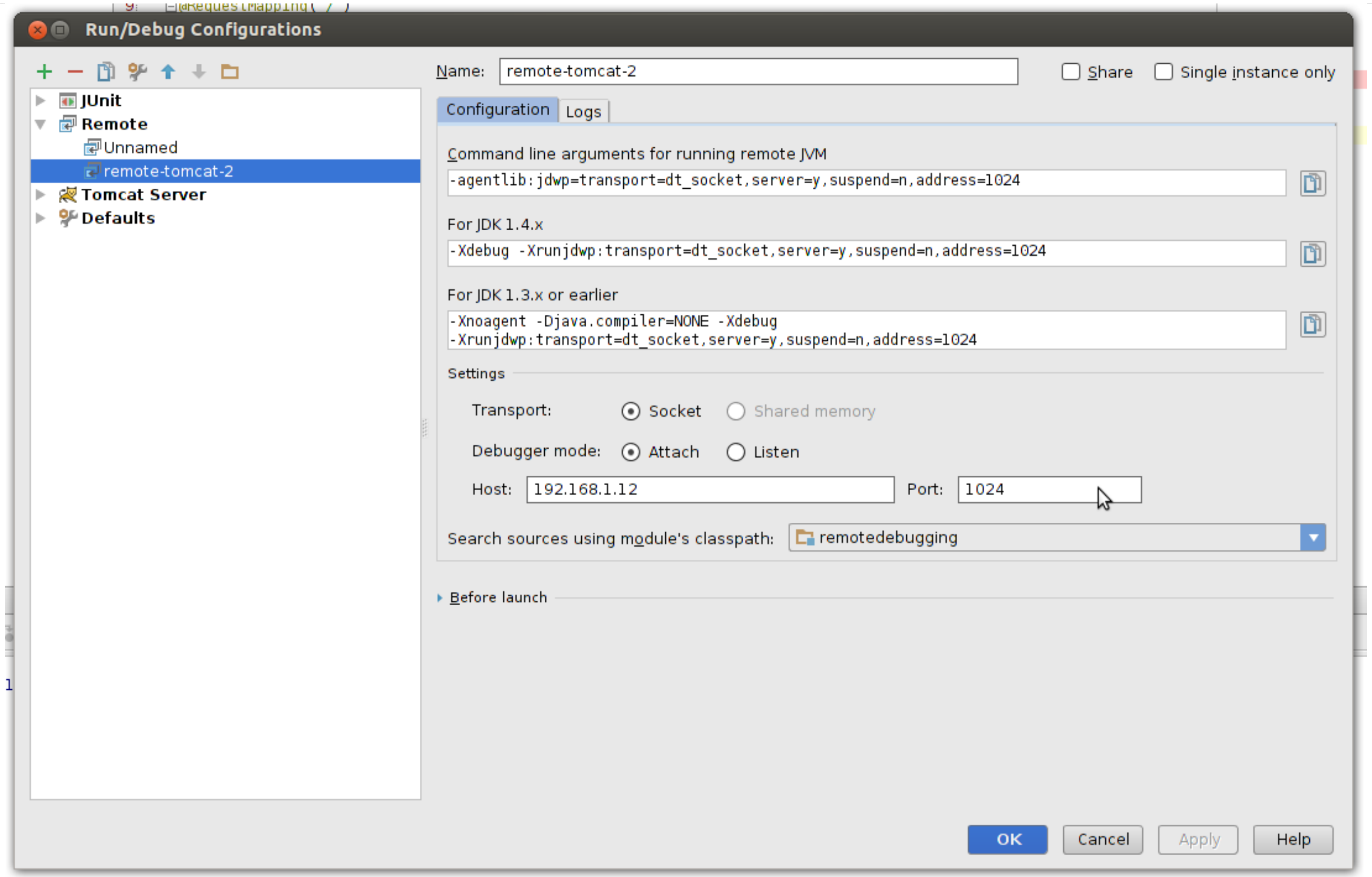
Using the Remote settings option

Open the Edit Configuration settings as done in the previous option but instead of selecting the Remote Tomcat, select the Remote option:





The Remote settings dialog box appears where you can specify the required configuration; remote host, port, project etc:



Name: remote-tomcat-2 [ ] Share [ ] Single instance only

Command line arguments for running remote JVM

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=1024
```

For JDK 1.4.x

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=1024
```

For JDK 1.3.x or earlier

```
-Xnoagent -Djava.compiler=NONE -Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=1024
```

Settings

Transport: [✓ Socket] Shared memory

Debugger mode: [✓ Attach] Listen

Host: 192.168.1.12 Port: 1024

Search sources using module's classpath: remotedebugging

Before launch

As you can see from above, the interesting thing about the Remote option is that, apart from allowing you to configure intelliJ IDEA, it also list the various configurations required to start up the remote JVM in other for remote debugging to work. Might come in handy.

Specify the required settings, click **Ok** to save changes, and start the debugging session. You should also see the notice that IntelliJ has successfully connected to the remote VM.

Once this is done, you should then open the source code of the application you have running on the remote Tomcat, put a breakpoint where required and you can go ahead and start debugging as if the application is running on your local machine.

So that is it with the configuration, the next question is, what makes all these fit and work together?

Let us take a quick work through of the technology at play that makes remote debugging JVM possible.

How Remote JVM Debugging Works

It all starts with what is referred to as Agents.

The JVM, which runs the complied `.class` sources has a feature that allows externally libraries (written in either Java or C++) to be injected into the JVM, just about runtime. These external libraries are referred to as Agents and they have the ability to modify the content of the .class files been run. These Agents have access to functionality of the JVM that is not accessible from within a regular Java code running inside the JVM and they can be used to do interesting stuff like injecting and modify the running source code, profiling etc. Tools like JRebel makes use of this piece of functionality to achieve their magic.

And to pass an Agent Lib to a JVM, you do so via start up arguments, using the -*agentlib:libname[=options]* format.

So in the configuration we had above:

```
-agentlib:jdwp=transport=dt_socket,address=1043,server=y,suspend=n
```

We were actually passing an Agent Lib named jdwp to the JVM running Tomcat. The jdwp is a JVM specific, optional implementation of the JDWP (Java Debug Wire Protocol) that is used for defining communication between a debugger and a running JVM. It's implementation, if present is supplied as a native library of the JVM as either jdwp.so or jdwp.dll

So what does it do?

In simple terms, the jdwp agent we pass is basically serving the function of being a link between the JVM instance running an application and a Debugger (which can be located either remote or local). Since it is an Agent Library, It does have the ability to intercept the running code, create a bridge between the JVM and a debugger, and have the functionality of a debugger applied on the JVM.

Since in the JVM architecture, the debugging functionality is not found within the JVM itself but is abstracted away into external tools (that are aptly referred to as debuggers), these tools can either reside on the local machine running the JVM being debugged or be run from am external machine. It is this de-coupled, modular architecture that allows us to have a JVM running on a remote machine and using the JDWP, have a remote debugger be able to communicate with it.

When IntelliJ IDEA was configured above, what was actually been done is to specify to the debugger tool within IntelliJ IDEA the host where the running JVM it needs to debug resides and the port through which it should connect with.

All the specification that outlines this modular architecture is contained in what is referred to as the Java Platform, Debugger Architecture, JPDA (this explains the JPDA in the JPDA_OPTS method used above) and you can read a much detailed overview of it here: Java Platform Debugger Architecture Overview.

Posted in: From The Trenches, Java

10 Responses

August 6, 2014 at 21:04 by jelmer Kuperus |

Out of the box you can already do

./catalina.sh jpda run

That will start tomcat in debug mode, listening for connections on port 8000

August 15, 2014 at 10:56 by Dadepo Aderemi |

Very true Jelmer, and worth noting. I have updated the post to include this option.

November 6, 2014 at 21:54 by arnski |

Thanks for this wellwritten and informative tutorial. Might come in handy developing on a mac using tomcat in a docker container.

November 30, 2014 at 16:24 by Stephane |

Hi,

I'm having trouble seeing the table of contents of the article. I can't make the parts out.

Other than that the article is informative.

Thanks.

January 26, 2015 at 22:30 by malloc4k |

Great and up to date tutorial. Thanks a lot !

March 16, 2015 at 10:23 by Heri |

Using your first option (JPDA_OPTS in setenv.sh) did not work immediately on tomcat 7.0.57.
Omitting the spaces in your example fiexed the problem:
export JPDA_OPTS="-agentlib:jdwp=transport=dt_socket,address=1043,server=y,suspend=n"

April 13, 2015 at 10:59 by Nitin Ramachandra |

@Dadepo Aderemi Amazing post… !!! Was very helpful.

April 27, 2015 at 14:59 by javabuh |

Thanks very much. Works like a charm. Great work!!

May 6, 2015 at 06:21 by Ronak |

hey I started the
Connected to the target VM, address: '10.1.123.100:8080', transport: 'socket'

Now how should I debug the code
Actually i want to do the following :
http://stackoverflow.com/questions/28436833/intellij-idea-run-scala-repl-console-on-a-remote-machine#

May 7, 2015 at 16:08 by davidjmcclelland |

Windows Service Java Tap arg method worked great- thank you

« Validating epubs

My Goto Amsterdam 2014 »

About this blog

The content on this blog is developed mostly by our own developers. It contains a selection of best practices, trends and technical information on different open source technologies. Our blog is where we can share our experiences with technologies we are experimenting with, advancing with and more often than not indulging in. In true open source community style, we don't hold back and we share code snippets, screen shots and provide insight wherever we can. We particularly

In many of our blogs, you'll find we endeavour to take you on a journey of our thought process. This means some of our blogs can often become epics and have a whole series associated with them. We think that most of our readers appreciate reading about our experiences on new products, releases, conferences and technologies and to be honest its what makes us tick and what we thrive on most. Every now and then we also invite partners, industry leaders and others around us as guest writers enjoy setting up a dialogue with our readers and often there is also a thread of question and answers in the blogs too. Our blogs are not however all about just the technical successes but also our less successful efforts too ;-)

or we share some interesting conversations we have had with them. Suggestions, input and feedback is always welcome, just drop us a note and we'd be happy to hear from you.