Anleitung zum Einrichten des Remote-Debuggers unter IntelliJ IDEA

Das Tutorenteam
Softwarepraktikum 2021

23. September 2021

Zusammenfassung

Im SoPra werden Sie Ihr Projekt auf verschiedene Arten testen müssen, insbesondere mit Unit- und Systemtests. Für Erstere ist debuggen einfach. Für Zweitere ist das ein gutes Stück komplizierter. Wie genau Sie Ihr Projekt während eines Systemtests sinnvoll debuggen, erklärt dieses Dokument.

Die Systemtests funktionieren so, dass sie für jeden einzelnen Test eine neue Instanz des Servers starten, mit den im Systemtest angegebenen Werten für Karte, Seed usw.

Insbesondere wird also der Server nicht über die IDE gestartet, sondern indirekt. Die kanonische Weise, ein auf andere Art gestartetes Softwareprojekt zu debuggen, ist das sogenannte *Remote Debugging*. Insbesondere in Java ist das recht einfach - wenn man dem Java-Programm beim Start die richtigen Kommandozeilenparameter mitgibt, so wartet es zu Beginn darauf, dass ein Debugger sich auf das Programm aufschaltet. In der IDE kann man so eine Aufschaltung durchführen und dann das Programm wie gewohnt debuggen.

Das Systemtest-Framework kennt bereits die richtigen Argumente. Wenn man seine Systemtests also debuggen möchte, ist es ausreichend, dem Systemtest-Framework den sogenannten *Debugger-Port* anzugeben. Das Framework setzt dann die richtigen Kommandozeilenparameter. Was man dabei dem Systemtest-Framework wie angeben muss, erklären wir in Abschnitt 1.

Danach muss man die IDE nur noch so konfigurieren, dass diese sich auch auf den laufenden Server aufschaltet. Wir erklären dies beispielhaft für IntelliJ, für andere IDEs ist das Vorgehen oft ähnlich. Dies passiert in Abschnitt 2.

Ein FAQ für häufige Fehler findet sich in Abschnitt 3. Weitere Fragen können im Forum gestellt werden.

1 Systemtest-Framework

1.1 Kommandozeilenparameter

Das Systemtest-Framework stellt die folgenden Kommadozeilenparameter zur Verfügung:

`-h`, `--help`	Dokumentation ähnlich dieser anzeigen
`-p` x, `--port` x	x ist der Port, über den der Systemtest mit dem Server kommuniziert. $1024 < x < 65536$. Dieses Argument ist notwendig.
`-j` f, `--jar` f	f ist der Pfad zur Jar-Datei des Servers. Dieses Argument ist notwendig.
`-t` t, `--timeout` t	t ist das Timeout in Sekunden. Es beschreibt, wie lange auf den Server gewartet wird. $t > 0$.
`--debug` p	p ist der Debug-Port. Wenn dieses Argument angegeben wird, wird der Systemtest im Debug-Modus gestartet und das Timeout auf ∞ gesetzt, unabhängig davon, was bei `--timeout` angegeben wurde. $1024 < p < 65536$.
`--json` f	f ist eine Datei, in die reingeschrieben wird, welche Systemtests bestanden wurden. Das wird auf dem Server verwendet, ist ansonsten nicht relevant.
`--name` r	r ist eine RegEx. Wenn gesetzt, werden nur die Systemtests, deren Name von dieser RegEx gematcht werden, ausgeführt. Mehr über RegExes hier.
`--all`	Führt alle Arten Systemtests aus.
`--system`	Führt die „System“-Systemtests aus.
`--daily`	Führt die „Daily“-Systemtests aus.
`--secret`	Führt die „Secret“-Systemtests aus.
`--mutant` i	Startet den Server im Mutanten-Modus, also mit Argument `--mutant` i.

To-Do

`--name` in `--regex` umbenennen.
`--debug` muss Timeout auf ∞ (im Code -1) setzen - ist zwar schon so dokumentiert, aber noch nicht so im Code

Für uns interessant ist das Argument `--debug`. Hier müssen wir den Port angeben, auf dem der Debugger warten soll. Dafür wird per Konvention der Port 5050 verwendet. Starten wir also unsere Systemtests mit dem zusätzlichen Kommandozeilenparameter `--debug 5050`, so können wir später den Remote Debugger auf den vom Framework gestarteten Server aufschalten.

Um genau zu sein, startet das Framework bei Angabe des `--debug` *port*-Kommandos den Server zusätzlich mit den folgenden JVM-Argumenten. Diese sagen dann eben

der JVM, dass sie einen Remote Debugger auf Port *port* erwarten zu hat. Insbesondere wartet die JVM auf den Debugger, bevor der Server ausgeführt wird.

`-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=`*port*

Neben dem Debug-Kommando sollte man mit dem `--name`-Kommando einen bestimmten Systemtest auswählen. Ansonsten werden alle Systemtests der Reihe nach ausgeführt. Da die aber alle mit dem Debug-Modus gestartet werden, und dann darauf warten, bis sich der Debugger verbinden, müsste man sonst alle Tests durchdebuggen, bis man bei dem ankommt, der tatsächlich kaputt ist.

To-Do

Angeben, wir man die Systemtests manuell startet

1.2 Einrichten in Gradle/IntelliJ

Damit wir die Systemtests nicht jedes mal umständlich manuell starten müssen, können wir uns das in IntelliJ oder Gradle einmal einprogrammieren, um danach die Systemtests auf Knopfdruck richtig zu starten.

To-Do

Abwarten, bis klar ist, wie Gradle aussieht, und basierend darauf diesen Abschnitt schreiben.

2 IntelliJ-Einrichtung

Hinweis

Die nachfolgenden Screenshots wurden mit Version 2021.2 (Build `#IU-212-4746.92`) der (für Studierende kostenlose) „Ultimate Edition“ von IntelliJ IDEA erstellt. In anderen Versionen können die Menueinträge leicht abweichende Namen haben, oder das Design verschieden sein.

In IntelliJ wollen wir nun eine neue *Run Configuration* hinzugen. Dazu drücken wir oben rechts auf die Drop-Down-Box. Eventuell erscheint dort keine Drop-Down-Box, sondern es steht da nur `Add Configuration...`, wie es in Abbildung 1 der Fall ist. Sofern sich das Fenster zum Einstellen der Run Configurations nicht direkt öffnet,

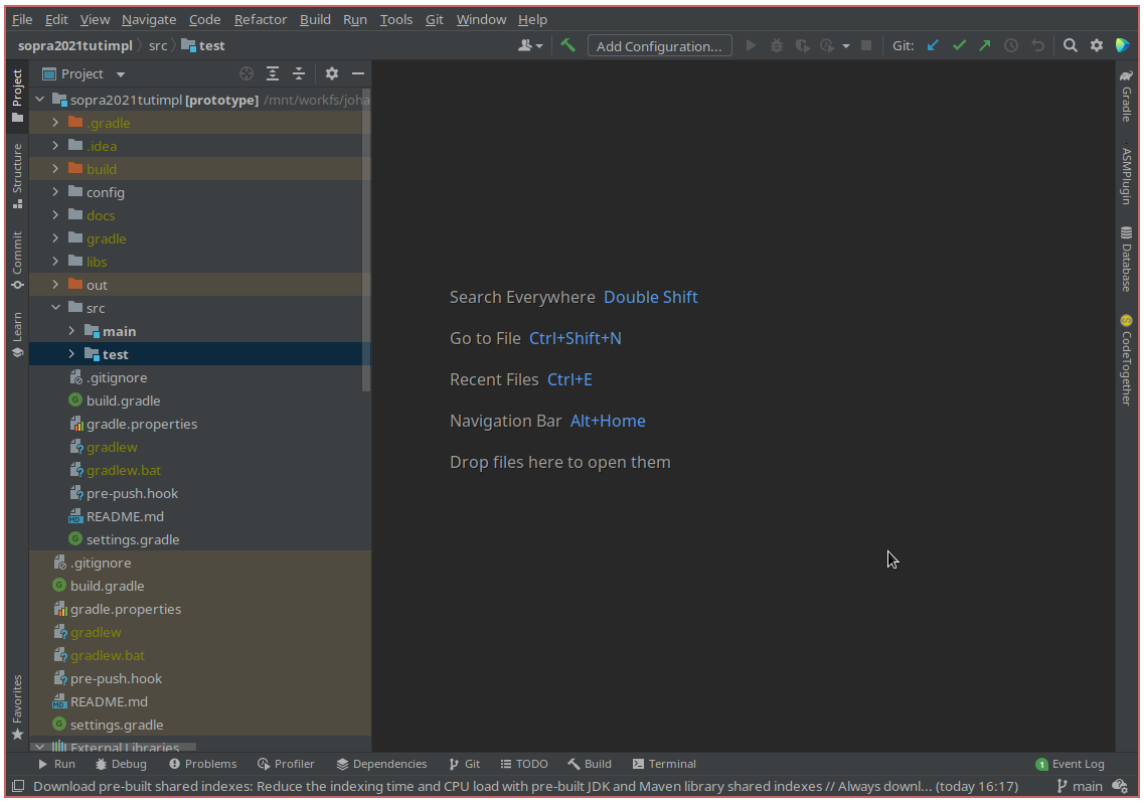

Abbildung 1: IntelliJ nach dem Öffnen. Oben rechts der Punkt `Add Configuration...`

wählen wir in Abbildung 2 den Menupunkt `Edit Configurations...`

Es öffnet sich nun das in Abbildung 3 gezeigte Fenster.

Dort klicken wir auf den +-Knopf oben links, um eine neue Konfiguration anzulegen, wie in Abbildung 4 gezeigt.

Nun erscheint eine Auswahlbox mit sehr vielen Option. Wir möchten die Option `Remote JVM Debug`. Diese findet man entweder, indem man weit genug nach unten

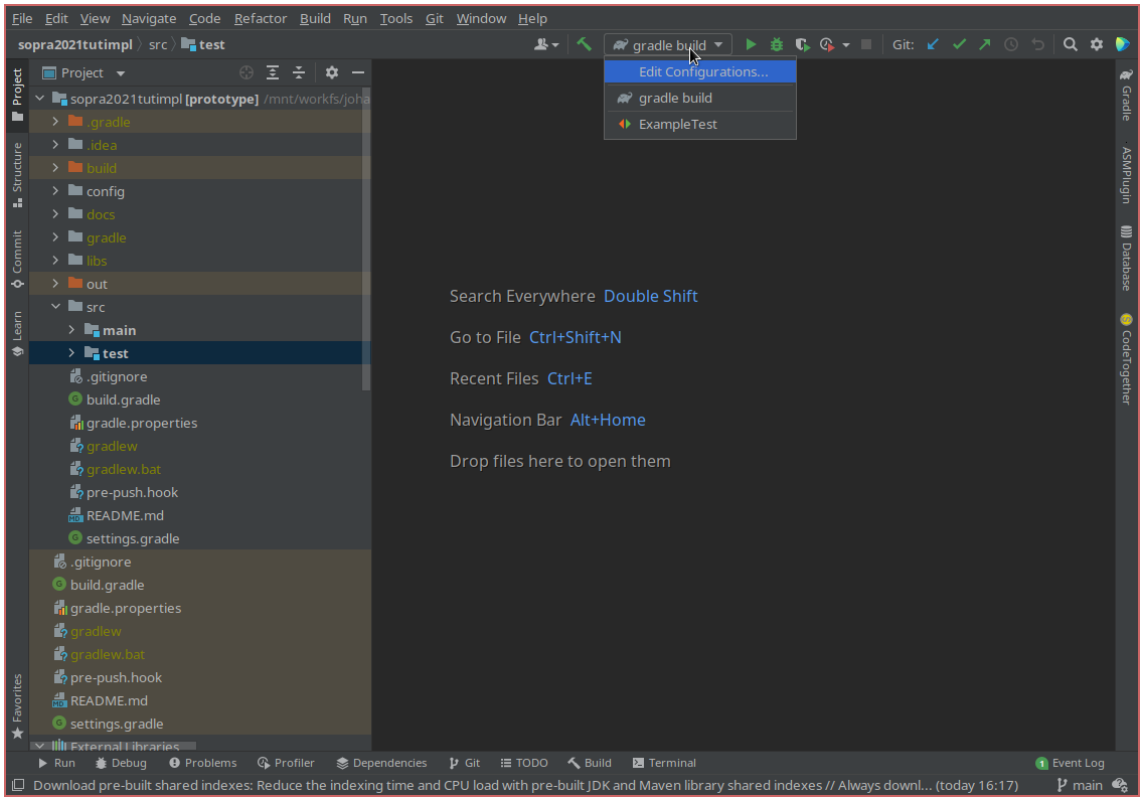

Abbildung 2: Die Auswahloptionen der Drop-Down-Box

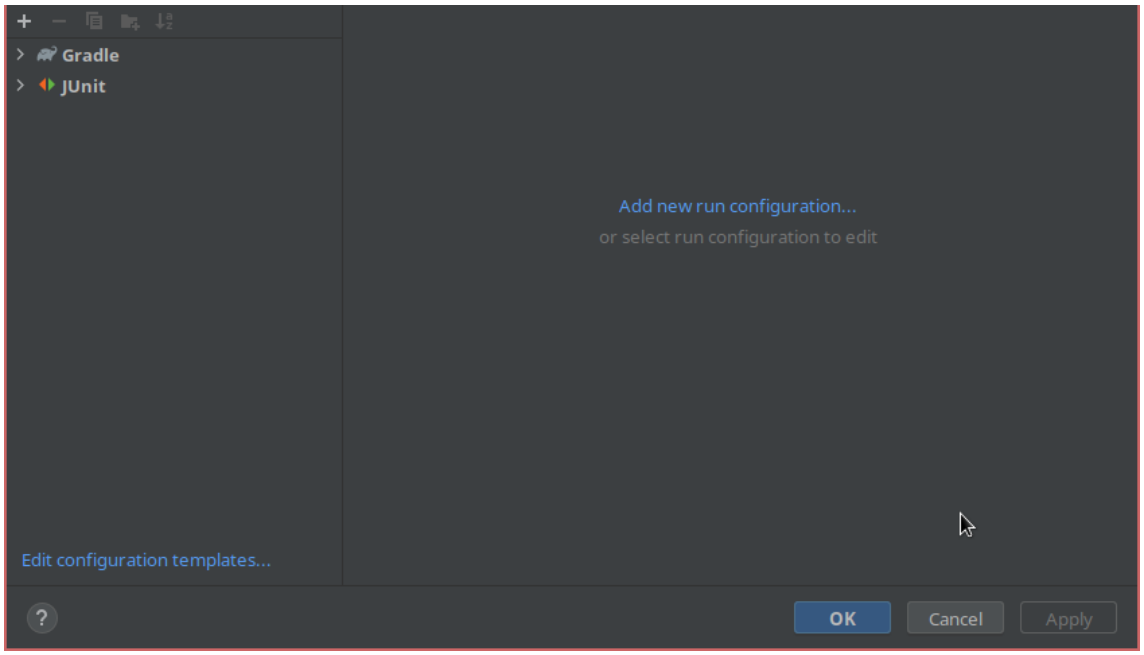

Abbildung 3: Das nun erscheinende Fenster „Run Configurations“

scrollt (die Optionen sind alphabetisch sortiert), oder indem man einfach anfängt, „Remote“ auf der Tastatur einzugeben, was dann eine Suche beginnt, wie in Abbildung 5 gezeigt.

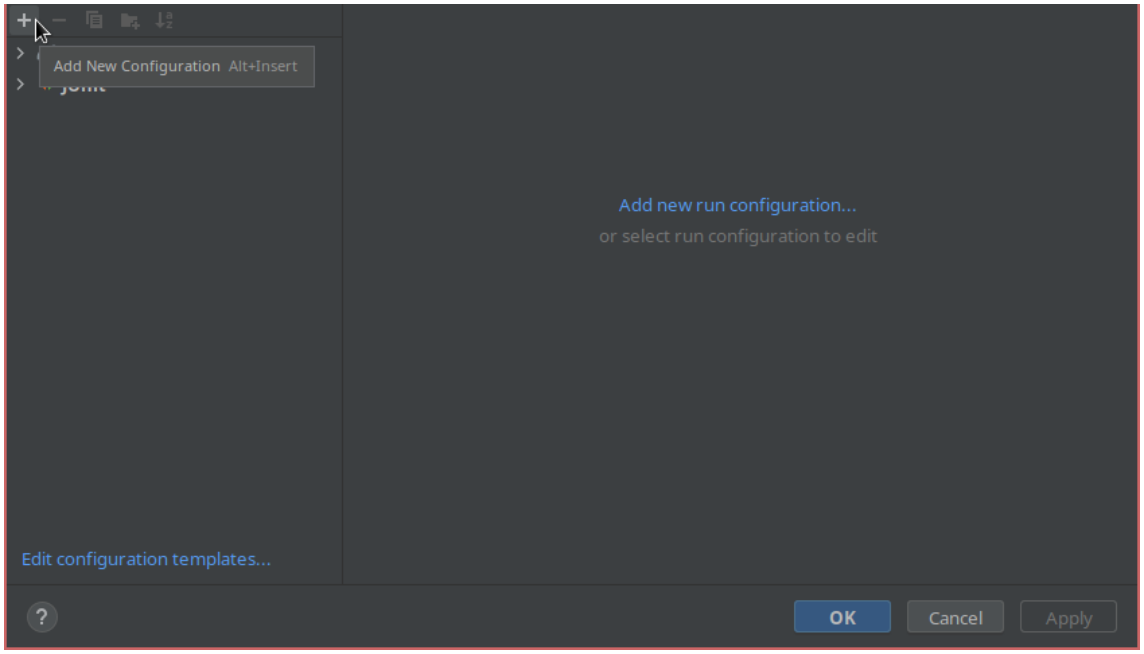

Abbildung 4: Der „Add New Configuration“-Knopf.

.

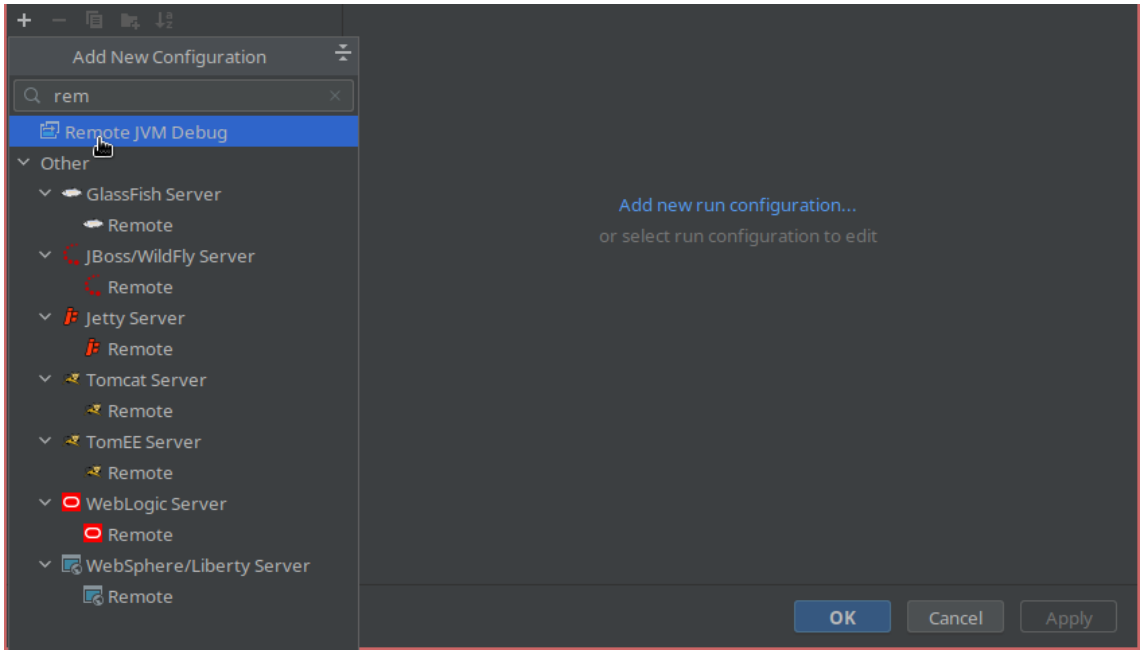

Abbildung 5: Suchen nach dem richtigen Typ

Wir haben nun eine neue Run Configuration mit dem richtigen Typ erstellt. Wir müssen diese jetzt noch feinjustieren. Zunächst ist es sinnvoll, einen passenden Namen zu vergeben. Dazu ändern wir einfach den Namen, wie Abbildung 6 zeigt.

Anschließend müssen wir bei `Use module classpath` den richtigen Classpath auswählen. Hier wählen wir den aus, der auf `.main` endet, wie Abbildung 7 zeigt.

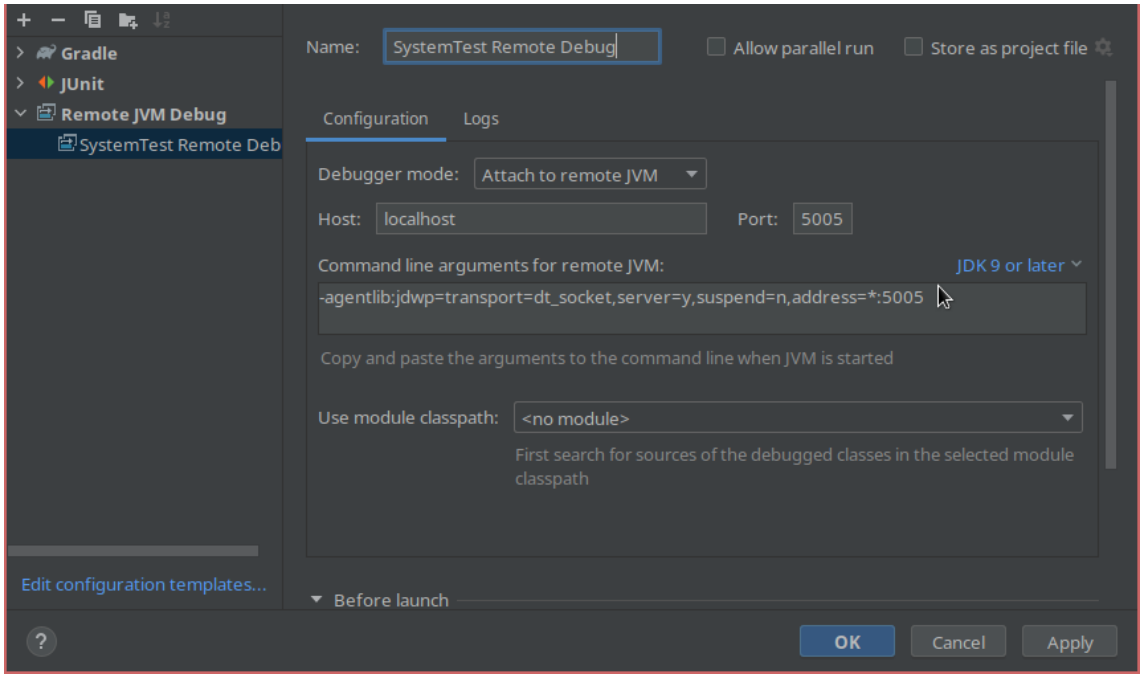

Abbildung 6: Setzen eines sinnvollen Namens

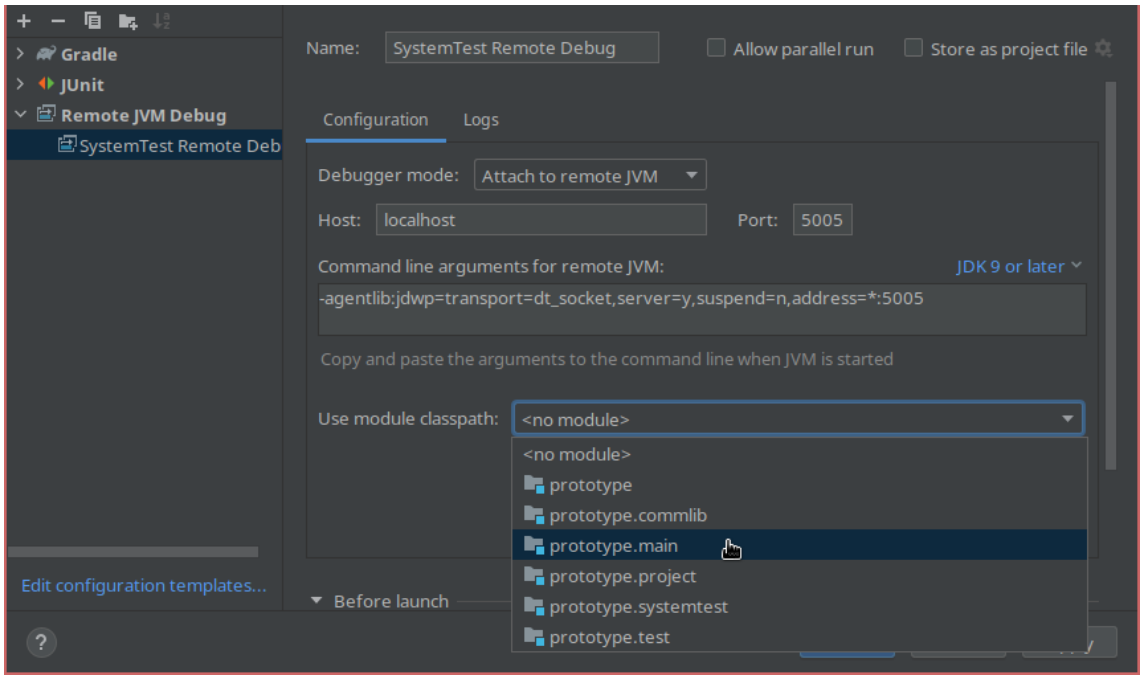

Abbildung 7: Setzen des verwendeten Classpaths

Zum Schluss kann man den Port ändern. Allerdings ist der Standartwert `5050` hier ausreichend. Man muss natürlich darauf achten, hier den gleichen Port zu wählen wie den, den man dann dem Systemtest-Manager mit dem Kommandozeilenparameter `--debug` *port* gibt (siehe Abschnitt 1.1). Wenn man ihn ändern muss, zeigt Abbildung 8, wie das geht.

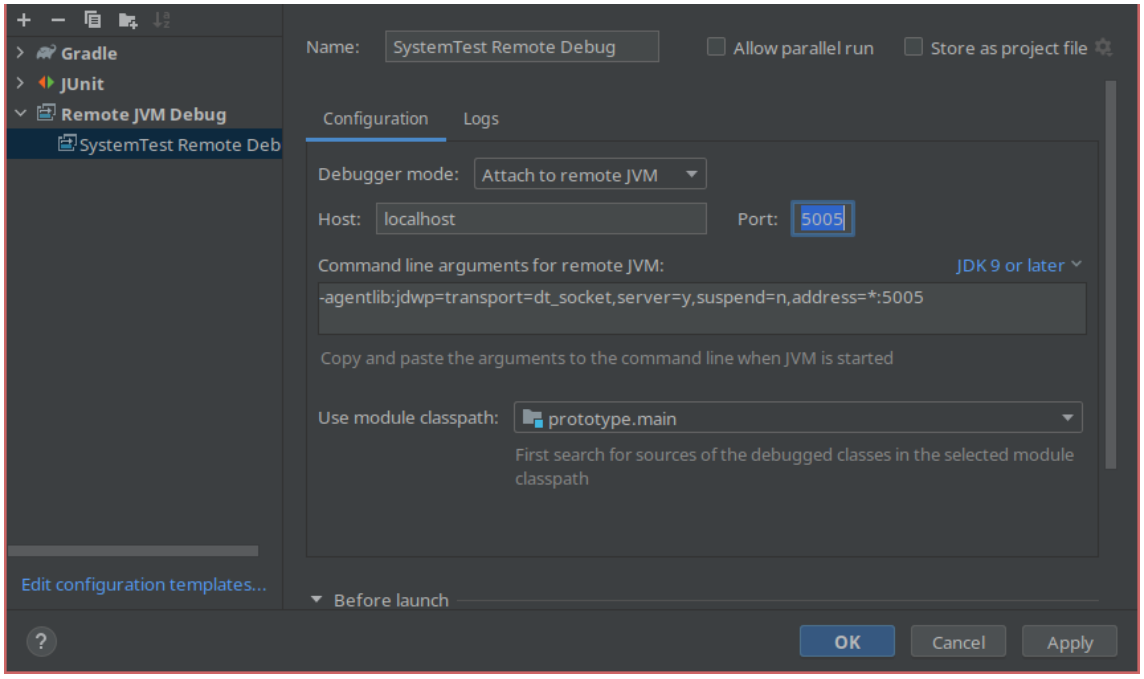

Abbildung 8: Das Feld, mit dem man den Port ändert

Jetzt sind wir fast fertig. Zunächst speichern wir diese Einstellungen und schließen das Einstellungsfenster mit einem Klick auf `Apply` oder `OK`, wie Abbildung 9 zeigt.

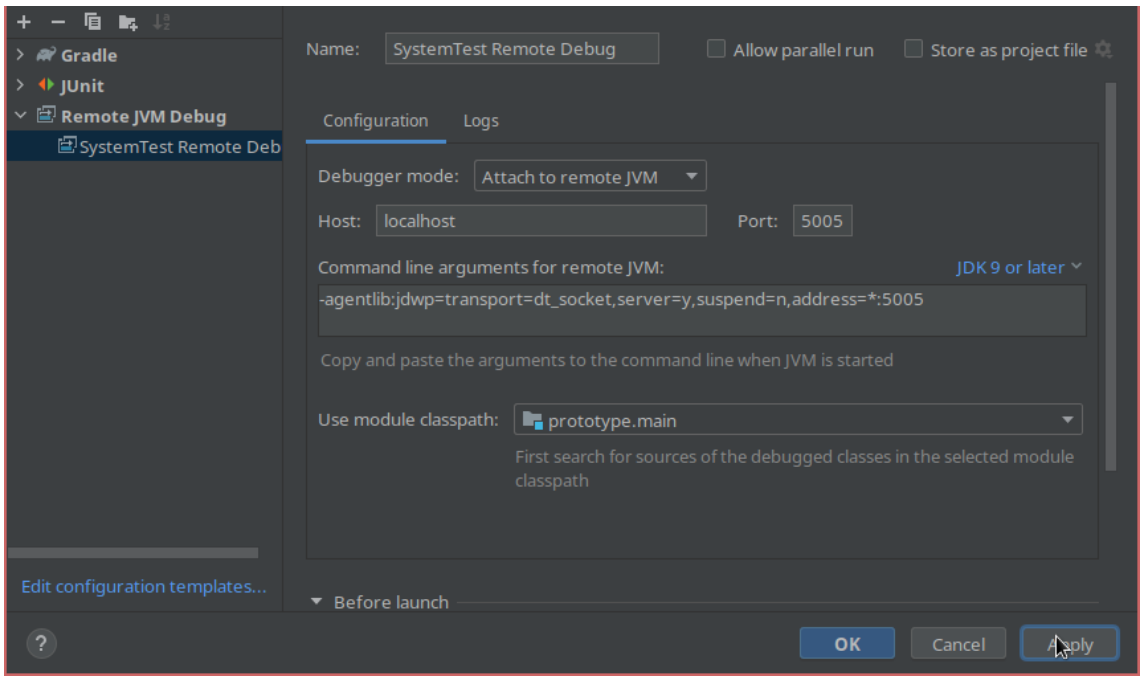

Abbildung 9: Klicken auf „Apply“

Um nun Ihr Projekt während der Ausführung eines Systemtests zu debuggen, müssen Sie zunächst die Systemtests wie in Abschnitt 1 beschrieben starten. Danach müssen

Sie nur noch die richtige Run Configuration auswählen, was Abbildung 10 zeigt, und dann auf den Debugging-Knopf drücken, wie in Abbildung 11 gezeigt.

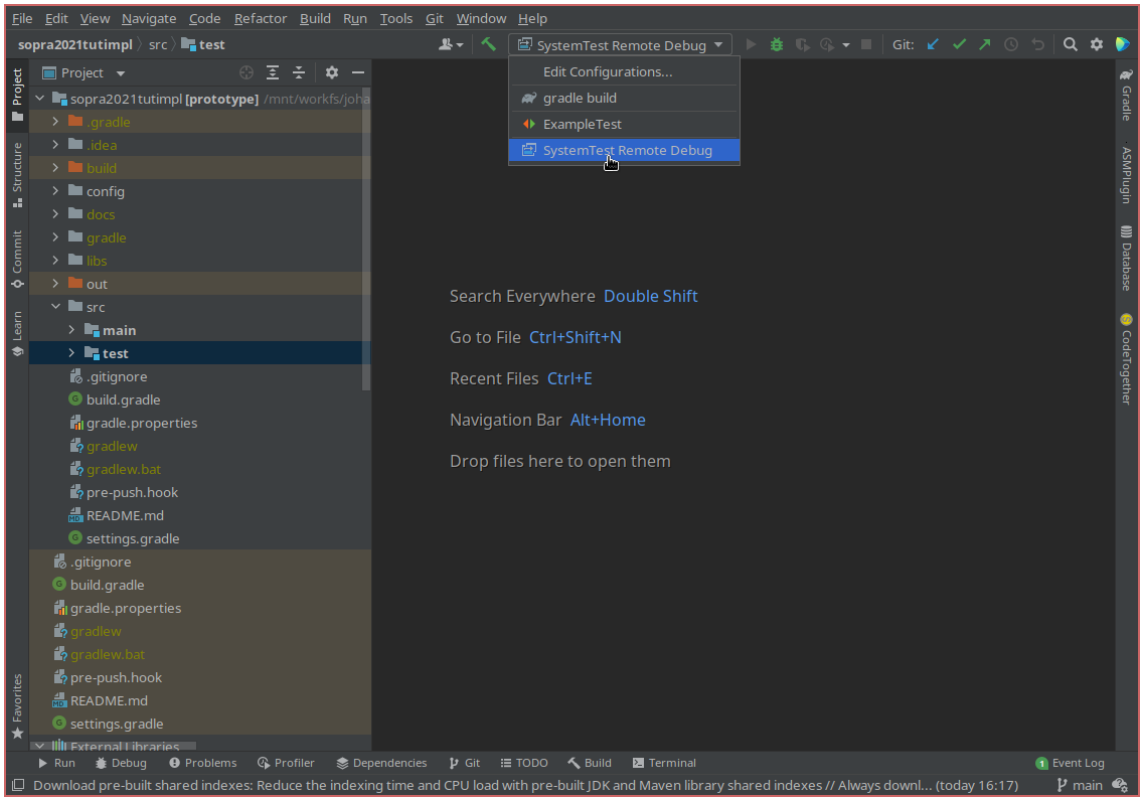

Abbildung 10: Auswahl der richtigen Run Configuration

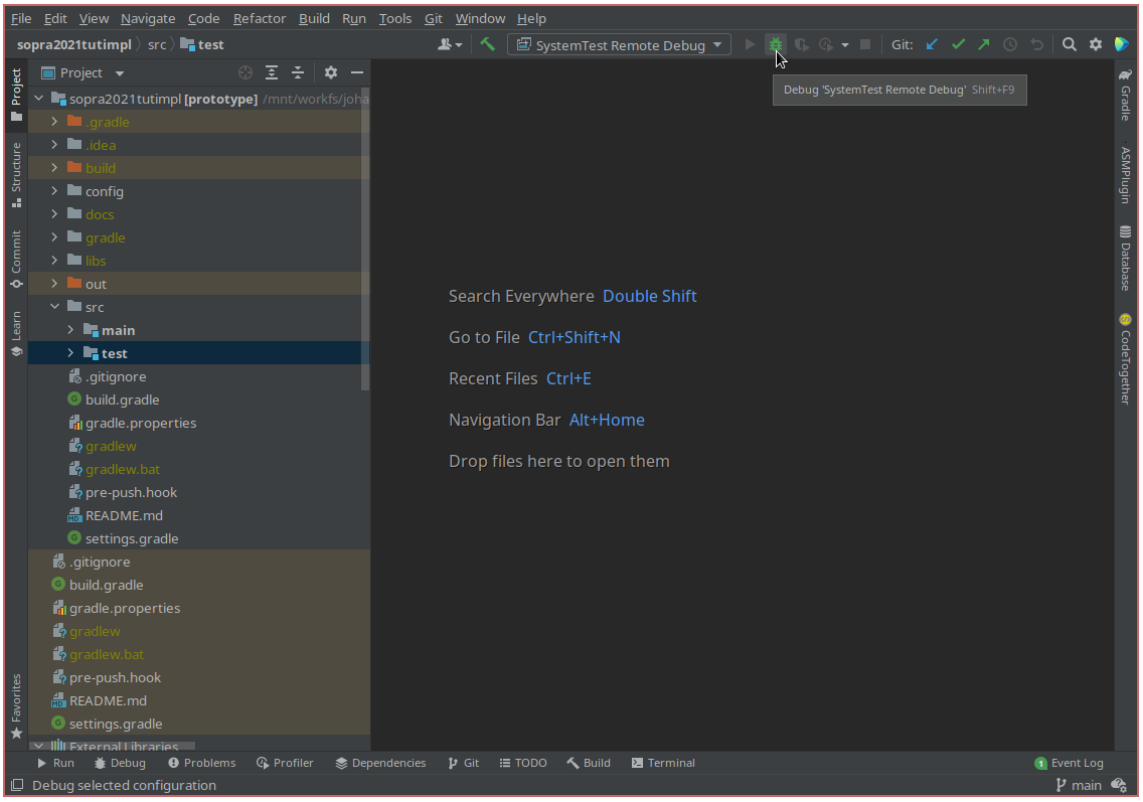

Abbildung 11: Starten des Debugging

3 FAQ

Wo stelle ich weitere Fragen?

Im Forum.