Debugging

Definition

Explain how to perform debugging when something doesn't work in Cargo

Sometimes, it can happen that the container does not start or stop as expected or you might have reasons to believe that CARGO is "acting weird". Here is a short list of things you can do to try debugging the problem.

Debugging the container

Most Java Virtual Machine implementations support remote debugging. Once started in debug mode, you can then remotely connect to your container using any IDE and debug your container and/or application.

In order to do so, add the following arguments to the JVM arguments configuration:

```
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspe
nd=<suspend>,address=<port>
-Xnoagent
-Djava.compiler=NONE
```

where:

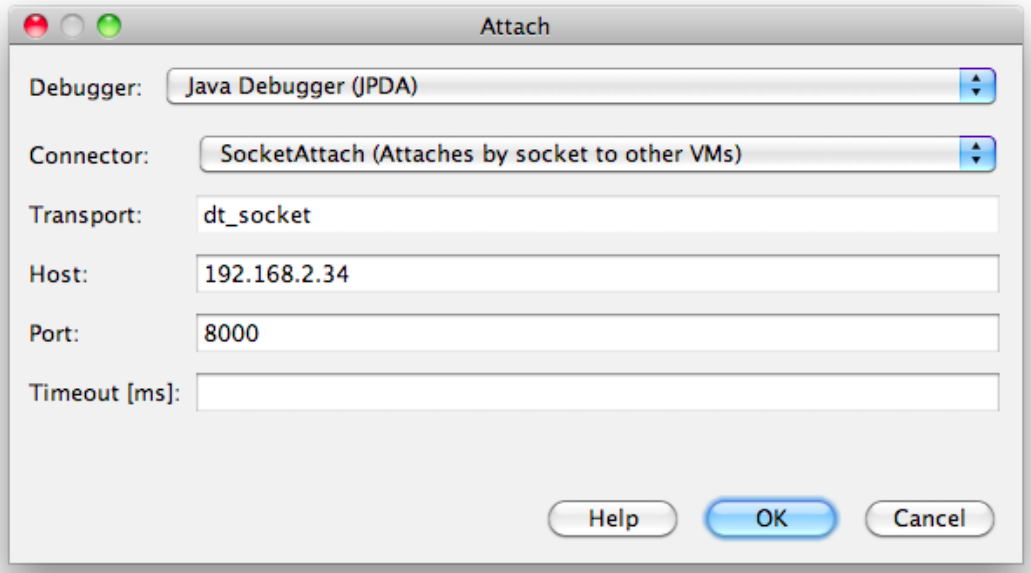
- **suspend** is whether the launch of the container should first wait for a remote debugger to connect. If y, the container will wait for you to connect with your remote debugger before starting; if n the container will start immediately and at the same time listen on the remote debugging port.
- **port** is the port number to use for remote debugging.

Here is an example for starting a JOnAS 5.x container in Remote Debug on port 8000 without suspend mode using the Maven2 plugin:

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>

<artifactId>cargo-maven2-plugin</artifactId>
  <version>${cargo.version}</version>
  <configuration>
    <container>
      <containerId>jonas5x</containerId>
      <type>installed</type>
      <home>${jonas.root}</home>
    </container>
    <configuration>
      <type>existing</type>
      <home>${jonas.base}</home>
      <properties>

<cargo.servlet.port>${http.port}</cargo.serv
let.port>
        <cargo.jvmargs>
          -Xdebug

-Xrunjdwp:transport=dt_socket,server=y,suspe
nd=n,address=8000
          -Xnoagent
          -Djava.compiler=NONE
        </cargo.jvmargs>
      </properties>
    </configuration>
  </configuration>
</plugin>
```

Once the server is started, follow these steps to remotely debug your server and/or application:

1. Click **Debug** - **Attach Debugger...**
2. In the window that appears, type in the remote host name and the port number then click OK



Steps for achieving the same in Eclipse IDE are similar.

Redirecting container output to a file

Example using the Java API

The `Container.setOutput(File)` API allows you to redirect the container console (`stdout`) to a file. This is the first file you should check in case of problem.

Cargo is able to configure containers to generate various levels logs. There are 3 levels defined in `o.c.c.container.property.LoggingLevel`: `LoggingLevel.LOW`, `LoggingLevel.MEDIUM` and `LoggingLevel.HIGH` (`LoggingLevel.MEDIUM` is the default value). They represent the quantity of information you wish to have in the generated log file. You can tune container logging by using the following API:

```
configuration.setProperty(GeneralPropertySet
.LOGGING, LoggingLevel.HIGH.getLevel());
```

The generated log files will then be found in the working directory you have specified on the container (through the `container.setWorkingDir()` call).

Starting Tomcat 4.x specifying an output console log file:

```
LocalContainer container = new
Tomcat4xLocalContainer(
    new
CatalinaStandaloneLocalConfiguration("target
/tomcat4x"));
container.setHome("c:/apps/jakarta-tomcat-4.
1.30");

container.setOutput("target/output.log");

container.start();
```

Use the `container.setAppend(true|false)` method to decide whether the log file is recreated or whether it is appended to, keeping the previous execution logs (by default, the file is recreated at every container start or stop).

Example using the Ant API

Starting Tomcat 4.x specifying an output console log file with the highest possible level of logs:

```
<cargo containerId="tomcat4x"
home="c:/apps/jakarta-tomcat-4.1.30"
    action="start"
    output="target/output.log"
    append="false">
  <configuration home="target/tomcat-home">
    <property name="cargo.logging"
value="high"/>
  </configuration>
</cargo>
```

Use the `append="true|false"` attribute for controlling the log file creation behavior.

Example using the Maven2/Maven3 plugin

Starting Tomcat 4.x specifying an output console log file with the highest possible level of logs:

```
<container>
  <containerId>tomcat4x</containerId>
  <home>c:/apps/jakarta-tomcat-4.1.30</home>
  <output>target/output.log</output>
  <append>false</append>
</container>
<configuration>
  <properties>
    <cargo.logging>high</cargo.logging>
  </properties>
</configuration>
```

Use the `append="true|false"` attribute for controlling the log file creation behavior.

Debugging CARGO itself

To enable the Java debugger on ANT, export the `ANT_OPTS` system property using the parameters specified above. For example:

```
export ANT_OPTS="-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspe
nd=y,address=8000"
```

On Maven2/Maven3, enabling Java debugger is done using the `mvnDebug` executable instead of `mvn`.

> **mvnDebug command on MacOS X**
>
> Note that on MacOS X, `mvnDebug` is not on the default path. To run it, execute the command: `/usr/share/maven/bin/mvnDebug`

Redirecting CARGO logs to a file

Example using the Java API

Some Cargo classes support generation of logs. This is implemented through the notion of `Logger` and levels are managed using `LogLevel`. There are 3 levels defined in `o.c.c.util.log.LogLevel`: `LogLevel.DEBUG`, `LogLevel.INFO` and `LogLevel.WARN` (`LogLevel.INFO` is the default value).

For example to turn on logging monitoring on a `Container` class, you can use:

```
Logger fileLogger = new FileLogger(new
File("c:/tmp/cargo.log"), true);
fileLogger.setLevel(LogLevel.DEBUG);
container.setLogger(fileLogger);
```

There are several Loggers that are readily available in the Cargo distribution:

- FileLogger: logs messages to a file
- SimpleLogger: logs messages to the console (stdout)
- AntLogger: logs messages using Ant's logging mechanism

Example using the Ant API

When using the Ant tasks, CARGO will by default use the AntLogger. You can specify the log level using the `logLevel` attribute, and the log file by using the `log` attribute. For example:

```
<cargo containerId="tomcat4x"
home="c:/apps/jakarta-tomcat-4.1.30"
    action="start"
    logLevel="debug"
    log="target/cargo.log"/>
```

Example using the Maven2/Maven3 plugin

When using the Maven2/Maven3 plugin, CARGO will by default use the MavenLogger.

When you use `mvn` with the `-e` option, both Maven itself and the CARGO plugin will log everything in debug mode.

If you want to use the plugin in debug mode without having Maven itself in debug mode, you can specify the log level using the `logLevel` attribute, and the log file by using the `log` attribute. For example:

```
<container>
  <containerId>tomcat4x</containerId>
  <home>c:/apps/jakarta-tomcat-4.1.30</home>
  <log>target/cargo.log</log>
  <logLevel>debug</logLevel>
</container>
```