About dl-check

DI-Check is a tool for finding potential deadlocks in Java programs via dynamic analysis approach.

GitHub Repository	https://github.com/Devexperts/dlcheck
Public Maven repo	https://bintray.com/devexperts/Maven/dl-check
License	GPLv3
Contact	dxlab@devexperts.com

DI-Check

DI-Check is a dynamic tool for finding potential deadlocks in multithreaded programs. It constructs the lock-order graph (similar to wait-for graph, but all added edges not being removed, so it reflect lock acquisitions history) and finds cycles in it. **DI-Check** uses various tecniques to get it fast and scalable and to avoid superfluous signals about potential deadlocks.

See this paper for details: DI-Check: Dynamic Potential Deadlock Detection Tool for Java Programs

How To

DI-Check is implemented as Java agent, so you should add -javaagent:dlcheck.jar option to analyze your application. To download the agent artifact use our Bintray repository: https://bintray.com/devexperts/Maven/dl-check.

Command Line

Here is an example of command line usage:

```
java -javaagent:dlcheck.jar -jar your_app.jar
```

Do not rename dlcheck. jar!

Maven

Use the following code in your pom.xml to use **DI-Check** for tests.

```
<!-- maven-dependency-plugin is used to
    copy "dlcheck" agent into target directory -->
<pluqin>
   <artifactId>maven-dependency-plugin</artifactId>
   <executions>
     <execution>
         <id>copy-sample-agent</id>
         <phase>process-test-classes</phase>
         <qoals>
            <goal>copy</goal>
         </goals>
         <configuration>
            <artifactItems>
               <artifactItem>
                  <groupId>com.devexperts.dlcheck</groupId>
                  <artifactId>agent</artifactId>
                  <version>${dlcheck.version}</version>
                  <outputDirectory>${project.build.directory}</outputDirectory>
                  <!-- Do not use another name! -->
                  <destFileName>dlcheck.jar</destFileName>
               </artifactItem>
            </artifactItems>
         </configuration>
      </execution>
  </executions>
</plugin>
<!-- Configure maven-surefire-plugin -->
<pluqin>
  <groupId>org.apache.maven.plugins/groupId>
   <artifactId>maven-surefire-plugin</artifactId>
   <configuration>
     <argLine>-javaagent:${project.build.directory}/dlcheck.jar
         -Ddlcheck.cache.dir=${dlcheck.workdir}/cache
         <!-- Optional, fails at the point of potential deadlock is detected -->
         -Ddlcheck.fail=true
         -Ddlcheck.output=${dlcheck.workdir}/potential_deadlocks
      </arqLine>
  </configuration>
</plugin>
<dependency>
  <groupId>com.devexperts.dlcheck</groupId>
   <art.ifact.Id>agent</art.ifact.Id>
  <version>${dlcheck.version}
  <scope>provided</scope>
</dependency>
```

Options

DI-Check can be configured via several system parameters (-Dparam.name=value).

- dlcheck.output defines path of file to be used for reporting. By default prints a report to the standard output.
- dlcheck.fail defines should DI-Check throws an exception at the point of potential deadlock detection. Disabled by default.
- dicheck.log.level defines internal logging level. Possible values: DEBUG, INFO (default value), WARN, ERROR.
- dicheck.log.file defines path of file to be used for logging. By default logs are printed to the standard output.
- dlcheck.cache.dir [experimental] defines directory to be used for transformed classes caching. This feature is unstable, use it on your own risk.
- dlcheck.include defines the transformation scope using globs. For example, setting the value to package.to.transform.*, another.package.to.transform.* informs DI-Check to transform classes from these packages only. By default all classes are included.
- dicheck.exclude defines the classes which should be excluded from transformation. The syntax is similar to dicheck.include option.

Output

Here is an example of DI-Check output. For each potential deadlock the associated cycle, currently acquired locks and stack trace are available.

```
!!! Potential deadlock !!!
-----
### Cycle in lock graph: ###
Lock Object@7ce69770 was acquired at:
       com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:37)
       com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:40)
Lock Object@7ce026d3 was acquired at:
       \verb|com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:36)| \\
       com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:41)
Edge 'Object@7ce026d3 -> Object@7ce69770' was added at:
       \verb|com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java: 41)| \\
       sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       \verb|sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)| \\
       \verb|sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)| \\
        java.lang.reflect.Method.invoke(Method.java:498)
       org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
       org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
### Current lock stack: ###
Lock Object@7ce69770 was acquired at:
       \verb|com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:37)| \\
       com.devexperts.dlcheck.tests.base.SynchronizedStatementTest.test(SynchronizedStatementTest.java:40)
```

Contacts

If you need help, you have a question, or you need further details on how to use DI-Check, you can refer to the following resources:

- dxLab research group at Devexperts
- GitHub issues

You can use the following e-mail to contact us directly: