[[RxJava] Debugging Tools](https://wiki.eisgroup.com/display/GRC/%5BRxJava%5D+Debugging+Tools)

The RxJava utility library provides a custom framework for debugging reactive code written using RxJava. A custom framework was implemented for debugging because:

All third party libraries target Java <= 8 and cannot use efficient StackWalker API

* Some of the libraries are no longer maintained and have inferior Rx code quality
* Some of the libraries that enrich the stack traces do so by wrapping the exceptions; this causes retry operator to no longer work
* Combined, this implementation adds **~6x** less overhead than RxJava2Debug (according to microbenchmarks), which is one of the most popular solutions for RxJava debugging

## **Debugging Tools**

### Invocation Logs

Allows logging of the operations performed on the reactive operators and schedulers. Operator log examples are provided below:

|  |
| --- |
| 17:00:35.686 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - ObservableOnErrorNext - subscribe, observer - TestObserver  17:00:35.688 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - ObservableError - subscribe, observer - OnErrorNextObserver  17:00:35.689 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - OnErrorNextObserver - onError, message - foo  17:00:35.690 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - ObservableJust - subscribe, observer - OnErrorNextObserver  17:00:35.691 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - OnErrorNextObserver - onNext, value - bar  17:00:35.691 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - TestObserver - onNext, value - bar  17:00:35.691 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - OnErrorNextObserver - onComplete  17:00:35.691 DEBUG [main] c.e.g.r.d.types.DebuggableObservable - TestObserver - onComplete |

### Stacktrace Enrichment

Allows enriching the stack trace of exceptions that occur in the reactive pipeline. An example exception is provided below:

|  |
| --- |
| java.lang.NullPointerException: The mapper returned a **null** item      at [[ ↓↓ Inferred construction point ↓↓ ]].(:0)      at io.reactivex.Maybe.map(Maybe.java:2974)      at com.eisgroup.genesis.rx.debug.RxDebugErrorTest.shouldEnrichNullMapperMaybe(RxDebugErrorTest.java:100)      at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)      at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)      at [[ ↓↓ Original trace ↓↓ ]].(:0)      at io.reactivex.internal.functions.ObjectHelper.requireNonNull(ObjectHelper.java:39)      at io.reactivex.internal.operators.maybe.MaybeMap$MapMaybeObserver.onSuccess(MaybeMap.java:82)      at com.eisgroup.genesis.rx.debug.types.DebuggableMaybe$DebuggableObserver.onSuccess(DebuggableMaybe.java:86)      at io.reactivex.internal.operators.maybe.MaybeJust.subscribeActual(MaybeJust.java:36)      at io.reactivex.Maybe.subscribe(Maybe.java:3726)      at com.eisgroup.genesis.rx.debug.types.DebuggableMaybe.lambda$subscribeActual$0(DebuggableMaybe.java:40)      at com.eisgroup.genesis.rx.debug.ConstructionBacktrace.withCurrent(ConstructionBacktrace.java:80)      at com.eisgroup.genesis.rx.debug.types.DebuggableMaybe.subscribeActual(DebuggableMaybe.java:40)      <...> |

Fatal errors are not enriched with construction stack traces due to the way RxJava handles them; such errors include instances of VirtualMachineError, ThreadDeath, and LinkageError

### Blocking Operation Logger

Warns the user about blocking operations which occurred in the reactive pipeline:

|  |
| --- |
| java.lang.RuntimeException: Detected a blocking call which can cause a livelock **if** no threads are available      at com.eisgroup.genesis.rx.debug.GenesisRxDebug.lambda$initBlockingLog$8(GenesisRxDebug.java:58)      at io.reactivex.plugins.RxJavaPlugins.onBeforeBlocking(RxJavaPlugins.java:1146)      at io.reactivex.internal.util.BlockingHelper.verifyNonBlocking(BlockingHelper.java:57)      at io.reactivex.internal.observers.BlockingMultiObserver.blockingGet(BlockingMultiObserver.java:82)      at io.reactivex.Single.blockingGet(Single.java:2156)      at io.reactivex.internal.operators.single.SingleFromCallable.subscribeActual(SingleFromCallable.java:35)      at io.reactivex.Single.subscribe(Single.java:2702)      at io.reactivex.Single.test(Single.java:3136)      at com.eisgroup.genesis.rx.debug.RxBlockingGetLogTest.shouldLogBlockingGet(RxBlockingGetLogTest.java:51)          <...> |

This feature implicitly enables RxJavaPlugins#setFailOnNonBlockingScheduler, which throws an exception if a blocking operation is called from a non-blocking scheduler

### RxJava Scheduler Watchdog

A watchdog that monitors the RxJava scheduler threads and notifies the user if some thread is stuck on a task. This feature helps in livelock detection. An example of such notification:

|  |
| --- |
| 16:59:01.156 WARN  [RxWatchdog-0] c.e.g.r.d.s.RxSchedulerWatchdog - Detected stuck task RxSingleScheduler-1:1814885044  16:59:01.159 WARN  [RxWatchdog-0] c.e.g.r.d.s.RxSchedulerWatchdog -  java.lang.RuntimeException: Stuck scheduler thread stack      at [[ ↓↓ Inferred construction point ↓↓ ]].(:0)      at io.reactivex.Completable.subscribeOn(Completable.java:1745)      at com.eisgroup.genesis.rx.debug.scheduler.RxSchedulerWatchdogTest.shouldAddStackToLog(RxSchedulerWatchdogTest.java:72)      at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)      at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)      at [[ ↓↓ Original trace ↓↓ ]].(:0)      at java.base@11.0.8/java.lang.Thread.sleep(Native Method)      at app//com.eisgroup.genesis.rx.debug.scheduler.RxSchedulerWatchdogTest.lambda$shouldAddStackToLog$2(RxSchedulerWatchdogTest.java:71)      at app//com.eisgroup.genesis.rx.debug.scheduler.RxSchedulerWatchdogTest$$Lambda$73/0x0000000800169440.run(Unknown Source)      at app//io.reactivex.internal.operators.completable.CompletableFromAction.subscribeActual(CompletableFromAction.java:34)      at app//io.reactivex.Completable.subscribe(Completable.java:1635)      at app//com.eisgroup.genesis.rx.debug.types.DebuggableCompletable.lambda$subscribeActual$0(DebuggableCompletable.java:40)      at app//com.eisgroup.genesis.rx.debug.types.DebuggableCompletable$$Lambda$81/0x000000080016e440.run(Unknown Source)      at app//com.eisgroup.genesis.rx.debug.ConstructionBacktrace.withCurrent(ConstructionBacktrace.java:80)      at app//com.eisgroup.genesis.rx.debug.types.DebuggableCompletable.subscribeActual(DebuggableCompletable.java:40)      at app//io.reactivex.Completable.subscribe(Completable.java:1635)      at app//io.reactivex.internal.operators.completable.CompletableSubscribeOn$SubscribeOnObserver.run(CompletableSubscribeOn.java:64)      <...> |

The watchdog works in tandem with stack trace enrichment. If the stack trace enrichment is enabled, the watchdog stack traces include the construction metadata, and if they are not, only the original trace is visible

## **Configuration**

The following flags can be used for configuring the debugging capabilities:

| **Property Name** | **Description** | **Default Value** |
| --- | --- | --- |
| genesis.rx.debug.full | Enables all debugging features, overrides other properties, except if they are explicitly disabled | **false** |
| genesis.rx.warn.blocking | Enables blocking operation logging | **false** |
| genesis.rx.invocation.log | A global flag that enables decorators for Rx schedulers and operators to log invocations | **false** |
| genesis.rx.capture.construction | Enriches Rx errors with reactive type construction information, enables construction stack capturing | **false** |
| genesis.rx.scheduler.watchdog | Enables logging of Rx threads stuck on a task | **false** |

The following properties can be used to configure the tool capabilities:

| **Property Name** | **Description** | **Default Value** |
| --- | --- | --- |
| genesis.rx.debug.stack.limit | The maximum number of stack elements captured (has an impact on performance) | **3** |
| genesis.rx.scheduler.watchdog.timeout.sec | The maximum number of seconds for a single task to be completed in the Rx scheduler | **30** |

If the  genesis.rx.invocation.log property is enabled, the following loggers can be configured using logback:

* **com.eisgroup.genesis.rx.debug.scheduler.RxLoggingScheduler** - for logging scheduler and worker operations
* **com.eisgroup.genesis.rx.debug.types.DebuggableCompletable** - for logging completable operations
* **com.eisgroup.genesis.rx.debug.types.DebuggableFlowable** - for logging flowable operations
* **com.eisgroup.genesis.rx.debug.types.DebuggableMaybe** - for logging maybe operations
* **com.eisgroup.genesis.rx.debug.types.DebuggableObservable** - for logging observable operations
* **com.eisgroup.genesis.rx.debug.types.DebuggableSingle** - for logging single operations

For the log messages to be visible, the loggers have to be set to DEBUG or TRACE level

Because they decorate the Rx operators, the debugging features can cause StackOverflow errors; the stack size must be increased in such cases