Coordination

Table of contents

Most scripts are written so that their worker threads operate independently of each other. For web load generation, a worker thread corresponds to the actions of a single, independent user. Worker threads can generate unique data using methods such as getProcessNumber() (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#getProcessNumber()) and getThreadNumber() (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#getThreadNumber()) . Coordination of activity within a worker process can use standard Java or Jython synchronisation APIs.

Occasionally a script needs to coordinate worker threads across multiple worker processes. The Grinder supports this requirement through a distributed synchronisation feature, *barriers*.

1 Barriers

A barrier (../../g3/script-javadoc/net/grinder/script/Barrier.html) is a pre-arranged synchronisation point at which worker threads will wait for each other. There can be many synchronisation points; each uses a unique barrier name.

Each worker thread that wants to participate in a synchronisation point should create a barrier with the given name using the ScriptContext (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#barrier(java.lang.String)) . The worker thread can wait for all other threads that have created barriers with a particular name by calling await (../../g3/script-javadoc/net/grinder/script/Barrier.html#await()) .

Barriers are usually created in the `TestRunner.__init__` constructor to ensure every worker thread has created its barriers before any of the threads try to wait for the barrier.

1.1 Sample script

```
from net.grinder.script.Grinder import grinder

class TestRunner:
  def __init__(self):
    # Each worker thread joins the barrier.
    self.phase1CompleteBarrier = grinder.barrier("Phase 1")

  def __call__(self):

    # ... Phase 1 actions.

    # Wait for all worker threads to reach this point before proceeding.
    self.phase1CompleteBarrier.await()

    # ... Further actions.
```

1.2 Barrier scope

Distributed barriers that allow coordination across worker processes require that the worker processes are started with the console.

Barriers are not shared across worker processes that are not started using the console, even if they are started by the same agent. In this case, each barrier will only provide coordination locally, between the worker threads of a worker process.

1.3 Barrier life cycle

A worker thread can reuse a barrier by calling await (../../g3/script-javadoc/net/grinder/script/Barrier.html#await()) again. The call will block until the other workers using barriers with the same name all call `await`.

A worker thread can wait for a limited time by using one of the versions of `await` that allow a timeout to be specified. If the timeout elapses, the barrier instance will be cancelled and become invalid. Other worker threads will no longer wait for the cancelled barrier. A new barrier can be created if required.

Worker threads can remove themselves from a synchronisation point by cancelling (../../g3/script-javadoc/net/grinder/script/Barrier.html#cancel()) a barrier directly.