Platform SDK Developer's Guide

Resources Releasing in an Application Container

5/11/2025

Resources Releasing in an Application Container

To improve performance, Platform SDK only releases internal resources (such as threads) after a slight delay so that they can be reused in the near future if beneficial.

This delay in releasing resources can lead to warnings about memory leaks from application containers like Tomcat, because Tomcat checks if all application threads are stopped immediately without taking into account that Platform SDK intentionally holds resources for a short time.

Starting with release 8.5.300.02, Platform SDK for Java includes a `PSDKRuntime` class that allows your applications to stop gracefully. When used in your application, this class does the following things:

- wait until all Platform SDK resources are released
- reduce time required to release PSDK resources

Design Notes

The `PSDKRuntime` class provides two methods:

- awaitTermination with timeout
- awaitTermination without timeout

PSDKRuntime.java

```
public final class PSDKRuntime {

  public static void awaitTermination() throws InterruptedException { ... } // It is similar
to call of awaitTermination(Long.MAX_VALUE, TimeUnit.MILLISECONDS)
  public static void awaitTermination(long timeout, TimeUnit timeUnit) throws
InterruptedException, TimeoutException { ... }

}
```

> **Important**
>
> These two methods wait until all Platform SDK resources are released, and reduce the release time for Platform SDK resources.

To initiate Platform SDK resources releasing:

- all Platform SDK channels have to be closed;

- if Platform SDK invokers were requested using `InvokerFactory.namedInvoker(String)` or `InvokerFactory.namedInvoker(String, int)`, then those invokers must be released (as many times as they were requested) using `InvokerFactory.releaseInvoker(String)`;
- if `SingleThreadInvoker` instances were created by user then these invokers have to be released using `SingleThreadInvoker.release()` ;
- if you scheduled timer actions by using `Scheduler.schedule(long, long, TimerAction)` then these timer actions have to be canceled using `TimerActionTicket.cancel()`.

Code Sample

The following sample provides an example of how you can correctly finalize a Platform SDK-based application in J2EE containers.

TestServlet.java

```
@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {

    ConfServerProtocol protocol;
    AsyncInvoker myInvoker;
    SingleThreadInvoker myInvoker2;
    TimerActionTicket ticket;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);

        String name = "Case0001731406Test";
        String host = "host";
        String clientName = "clientName";
        String userName = "userName";
        String password = "password";
        int port = 2020;

        // creates PSDK channel
        protocol = new ConfServerProtocol(new Endpoint(name, host, port));
        protocol.setClientName(clientName);
        protocol.setUserName(userName);
        protocol.setUserPassword(password);
        protocol.setTimeout(Long.MAX_VALUE);

        // request PSDK named invoker
        myInvoker = InvokerFactory.namedInvoker("myInvoker");
        protocol.setInvoker(myInvoker);

        // request it 2nd time (it increases its reference counter)
        InvokerFactory.namedInvoker("myInvoker");

        // request PSDK invoker
        myInvoker2 = new SingleThreadInvoker();

        // open PSDK channel
        try {
            protocol.open();
        } catch (Exception e) { /*...*/ }

        // creates PSDK timer action
```

```
        TimerAction action = new TimerAction() {
            public void onTimer() {   /* ... */  }
        };

        // schedule the periodic timer action
        ticket = TimerFactory.getTimer().schedule(500, 1000, action);
    }

    public void destroy() {

        // stop periodic timer action scheduled before in PSDK timer
        if (ticket != null) {
            ticket.cancel();
            ticket = null;
        }

        // close all opened PSDK channels
        if (protocol != null) {
            try {
                protocol.close();
            } catch (Exception e) { /*...*/ }
            protocol.setInvoker(null);
            protocol = null;
        }

        // release 1st invoker
        if (myInvoker != null) {
            myInvoker = null;
            InvokerFactory.releaseInvoker("myInvoker");
            InvokerFactory.releaseInvoker("myInvoker"); // named invoker have to be released
as many times as it was requested before
        }

        // release 2nd invoker
        if (myInvoker2 != null) {
            myInvoker2.release();
            myInvoker2 = null;
        }

        // wait until all PSDK resources are stopped
        try {
            PSDKRuntime.awaitTermination(20000, TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) { /*...*/ }
    }


    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException { /* ... */ }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException { /* ... */ }

    // ...
}
```