

Base Language - Feature #6424

fill gaps in Java Open Client replacement (appserver client support)

05/26/2022 04:55 PM - Greg Shah

Status:	WIP	Start date:	
Priority:	Normal	Due date:	
Assignee:	Galya B	% Done:	0%
Category:		Estimated time:	0.00 hour
Target version:		version:	
billable:	No		
vendor_id:	GCD		
Description			
Related issues:			
Related to Base Language - Support #6488: write tests which use Java Open Cli...			New
Related to Runtime Infrastructure - Feature #6673: rework appserver agent app...			New

History

#1 - 05/26/2022 04:56 PM - Greg Shah

- Review the OE classes for their Java Open Client and add helper methods to LegacyJavaAppserverClient as needed to functionally replace anything missing.
- Investigate 'cancel request' and multi-threading for the remote appserver.

#3 - 01/12/2024 06:29 AM - Galya B

Are there any examples how to setup a java client in FWD and in OE?

#4 - 01/12/2024 08:23 AM - Constantin Asofiei

I have a Java project which shows how to execute OpenClient calls in both FWD and OE, I'll make it available in the next few days.

#5 - 01/31/2024 07:17 AM - Galya B

- Related to Support #6488: write tests which use Java Open Client to access the appserver testcases added

#6 - 03/22/2024 09:22 AM - Galya B

Constantin, where is the project on the shared drives? I remember you gave me a path, I can't find now.

#7 - 03/22/2024 09:24 AM - Galya B

Nevermind, it was on devsrv01: scp devsrv01:/tmp/gradle-skylab.zip . .

#8 - 03/22/2024 09:37 AM - Galya B

This will need a guide - how to use gradle cache to run and debug a Java client project.

#9 - 04/01/2024 04:00 AM - Constantin Asofiei

Galya, I've placed the project under devsrv01:/tmp/open_client_example.zip. There is a small readme.txt in the project root.

This has examples on how to use OpenClient from both FWD and OpenEdge. Please post any questions you have.

Thanks.

#10 - 04/17/2024 09:36 AM - Galya B

- File *FWDDTest-result.log* added

There are some failing tests. Is it the deployment or it's fine?

#11 - 04/18/2024 06:44 AM - Galya B

- File *FWDDTest-server.log* added

This is the server log of errors.

#12 - 04/30/2024 08:19 AM - Galya B

I think we're doing a partial review of the Java client in #8456 and #8661. At least I'm getting familiar with the client code. The test project is less more complicated than the customer project I'm reviewing.

I wonder what should be the focus of this task.

#13 - 04/30/2024 09:12 AM - Galya B

- Status changed from New to WIP

- Assignee set to Galya B

There is one point in `AppServerConnectionPool` that is bothering me. It's used only by `LegacyJavaAppserverClient` and `LegacyServiceHandler`:

```
/** The pool of workers dedicated to executing legacy services. */
private PriorityQueue<LegacyServiceWorker> workers = new PriorityQueue<>();

/** Mapping of workers by their connection ID. */
private Map<String, LegacyServiceWorker> workersByConnection = new ConcurrentHashMap<>();

public boolean dispatch(String connectionID, Consumer<LegacyServiceWorker> work, String target)
{
    // TODO: increase the workers based on the appserver settings (max_agents, etc)????

    // find a worker
    LegacyServiceWorker worker;
    if (connectionID == null)
    {
        try
        {
            worker = workers.take();
        }
        catch (InterruptedException e1)
        {
            // work interrupted
            LOG.log(Level.SEVERE, "Interrupted worker for " + target);

            return false;
        }
    }
    else
    {
        worker = workersByConnection.get(connectionID);

        if (worker == null)
        {
            // work interrupted
            LOG.log(Level.SEVERE,
                "Could not find worker for connection " + connectionID + " when processing " + target);

            return false;
        }
    }

    Throwable[] exc = new Throwable[1];
```

```

CountDownLatch down = new CountDownLatch(1);
worker.addWork(() ->
{
    try
    {
        // this ensures the queue remains sorted...
        workers.remove(worker);
        workers.add(worker);

        work.accept(worker);
    }
    catch (Throwable t)
    {
        exc[0] = t;
    }
    finally
    {
        down.countDown();
    }
});

boolean done = false;
try
{
    if (timeout == 0)
    {
        down.await();
        done = true;
    }
    else
    {
        done = down.await(timeout, TimeUnit.SECONDS);
    }
}
catch (InterruptedException e)
{
    // work interrupted
    LOG.log(Level.SEVERE, "Interrupted worker for " + target);

    return false;
}
finally
{
    // this ensures the queue remains sorted...
    workers.remove(worker);
    workers.add(worker);
}

if (exc[0] != null)
{
    if (exc[0] instanceof RuntimeException)
    {
        throw (RuntimeException) exc[0];
    }

    throw new RuntimeException(exc[0]);
}

if (!done)
{
    LOG.log(Level.SEVERE, type + " call " + target + " terminated, timeout elapsed.");
    return false;
}

return true;
}

```

I think the queue workers doesn't stay sorted as expected. The calls to dispatch sometimes have connectionId and sometimes not, so sometimes the if block is executed, other times the else. When there is a connectionId the taken worker is not the next in the queue, but the logic below still applies.

addWork adds a task. When the task starts execution the worker it uses gets removed from its place in the queue (even if it's not the next element)

and added back as last. The same operation is repeated on task execution completed (in the finally block). This method is not synchronized, also tasks will be executed by different workers in different threads, so remove-add is not one atomic operation.

`workers.take()` removes a worker, but adds it back right before the execution of the task starts.

I would suggest to remove the remove/add from both places:

```
workers.remove(worker);  
workers.add(worker);
```

Then add `workers.add(worker)` right after `worker = workers.take();`. This way the queue will always be sorted. `workers.take()` will never block for long and will work as the else case, where no blocking operation is present, in other words the tasks will be immediately added to the loop / queue in the worker itself.

#15 - 05/01/2024 01:33 AM - Galya B

- Related to Feature #6673: rework appserver agent approach to use a shared queue instead of a check-in/check-out registry added

#17 - 05/03/2024 07:40 AM - Constantin Asofiei

Galya B wrote:

There are some failing tests. Is it the deployment or it's fine?

The failing tests are known failures at this time.

The errors in the server log - I can't say how/where they are related.

About the queue question:

- the OpenClient first needs to establish a connection - this means it gets a worker from the queue, and saves that connection on the OpenClient side. This is the case where `take()` is called.
- future calls will always be done via this worker.
- a persistent procedure call ran on this worker will be posted to the worker's queue, and will not bypass this
- in Session-free, the binding is **not** done on the connection, you can have multiple persistent procedures ran on the same connection, each binding on a different agent's session (or actual agent in classic appserver)
- the workers use a `PriorityBlockingQueue` which is thread-safe. Also, the add/remove is meant to re-sort the queue so that the workers with the smallest amount of pending tasks are in front of the queue (which will be picked up when another OpenClient connection is established). Plus `take()` will block until a worker is available - **this does not mean that an FWD agent is available to get the request.**

So, the point here: there will **always** be a worker available to OpenClient, but is not guaranteed that an agent will be available to get and execute the task.

Files

FWDTest-result.log	3.09 KB	04/17/2024	Galya B
FWDTest-server.log	9.44 KB	04/18/2024	Galya B