

Jakarta Concurrency Guide

JEUS 9

TMAXSOFT

Copyright

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

Company Information

TmaxSoft Co., Ltd.

TmaxSoft Tower 10F, 45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, South Korea

Website: <https://www.tmaxsoft.com/en/>

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

Java, Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Microsoft, Windows, Windows NT are registered trademarks or trademarks of Microsoft Corporation.

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

AIX is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of X/Open Company, Ltd.

Linux is a registered trademark of Linus Torvalds.

Noto is a trademark of Google Inc. Noto fonts are open source. All Noto fonts are published under the SIL Open Font License, Version 1.1. (<https://www.google.com/get/noto/>)

Other products and company names are trademarks or registered trademarks of their respective

owners.

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (™, ®).

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory:
\${INSTALL_PATH}/license/oss_licenses

Document History

Product Version	Guide Version	Date	Remarks
JEUS 9	3.1.2	2025-03-24	-
JEUS 9	3.1.1	2024-12-24	-

Contents

1. Jakarta Concurrency	1
1.1. Overview	1
1.2. Managed Task	1
1.3. Container Thread Context	1
2. Managed Objects	3
2.1. ManagedExecutorService	3
2.2. ManagedScheduledExecutorService	5
2.3. ContextService	7
2.4. ManagedThreadFactory	9

1. Jakarta Concurrency

This chapter describes the background to the introduction of Jakarta Concurrency and its functionality.

1.1. Overview

It is recommended that application servers do not use concurrency APIs such as Thread, Timer, and ExecutorService provided in Java SE in the EJB or a web component. It is assumed that Jakarta EE application components such as Servlets and EJB are executed in a thread managed by an application server. It is also assumed that the functions provided in a container are executed in the same thread.

For these reasons, application components cannot safely use Jakarta EE services in a thread that is not managed by a container. Furthermore, if resources are used in a thread that is not managed by a container, potential issues regarding usability, security, reliability, and scalability can occur in Jakarta EE.

To solve problems that occur in unmanaged threads, Jakarta Concurrency specification, which extended the concurrency utilities of Java SE, is provided.

This technology guarantees the execution of applications without damaging the integrity of containers in the Jakarta EE environment.

1.2. Managed Task

In order to reduce consumption of unnecessary resources, a container must pool resources and manage life cycles. However, if asynchronous tasks are executed in a component by using the concurrency APIs provided in Java SE, then the container cannot manage the resources as it cannot recognize them.

As a result, this specification has defined managed tasks by expanding on the tasks defined in `java.util.concurrent` in the existing Java SE. This allows a container to manage general tasks as managed tasks, and maintains the execution context when a task is executed asynchronously.

1.3. Container Thread Context

In the Jakarta EE environment, a container contains the context information of each service when a service is executed.

If the Java SE's concurrency API is used in a component, then to maintain the context information of a service in a thread where a new container has been created, application developers must propagate the context in the following method.

1. Store the container context of the application component thread.

2. Determine which container context to store and propagate.
3. Apply the container context in the newly created thread.
4. Restore the context of the original component thread.

By using the above method, a task can be executed while maintaining the context of Java SE's concurrency API. However, if the Jakarta Concurrency services is used, user-defined tasks can be easily sent to managed objects. This allows the context to be automatically maintained and restored.

When Executing a Task by Maintaining the Context

- `java.util.concurrent.Callable`
 - `call()`
- `java.lang.Runnable`
 - `run()`
- `jakarta.enterprise.concurrent.ManagedTaskListener`
 - `taskAborted`
 - `taskSubmitted`
 - `taskStarting`
- `jakarta.enterprise.concurrent.ManagedTaskListener`
 - `taskAborted`
 - `taskSubmitted`
 - `taskStarting`
- `jakarta.enterprise.concurrent.Trigger`
 - `getNextRuntime()`
 - `skipRun()`

2. Managed Objects

This chapter describes the managed objects provided by Jakarta Concurrency by using examples.

2.1. ManagedExecutorService

The `jakarta.enterprise.concurrent.ManagedExecutorService` interface inherits Java SE's `java.util.concurrent.ExecutorService`. Just like `ExecutorService`, it is used for executing asynchronous tasks, and the application server maintains the context of the tasks that have been executed asynchronously.

Resource Definition Example

The following is an example of defining `ManagedExecutorService` as a resource.

Example of Defining `ManagedExecutorService` as a Resource: `<domain.xml>`

```
<domain>
  ...
  <server>
    <data-sources>
      <data-source>testdb</data-source>
    </data-sources>
    <managed-executor-service>mes1</managed-executor-service>
  </server>

  <resources>
    <managed-executor-service>
      <export-name>mes1</export-name>
      <long-running-task>true</long-running-task>
      <thread-pool>
        <min>10</min>
        <max>20</max>
        <keep-alive-time>60000</keep-alive-time>
        <queue-size>4096</queue-size>
        <stuck-thread-handling>
          <max-stuck-thread-time>3600000</max-stuck-thread-time>
          <action-on-stuck-thread>None</action-on-stuck-thread>
          <stuck-thread-check-period>300000</stuck-thread-check-period>
        </stuck-thread-handling>
      </thread-pool>
    </managed-executor-service>
  </resources>
  ...
</domain>
```

The configuration items are as follows.

- Basic Items

Item	Description
Export Name	Sets the name that will be used to register a managed executor service with the naming server.
Long Running Task	Indicates whether a task run by a managed executor service is long-running. Boolean type.

- Thread Pool

Configures the thread pool used in a managed executor service.

Item	Description
Min	Minimum number of threads managed by a thread pool.
Max	Maximum number of threads managed by a thread pool.
Keep Alive Time	Keep-alive time for inactive threads. If a thread pool contains more than the minimum number of threads, inactive threads for the specified time will be automatically removed. If this option is set to 0, threads will not be removed.
Queue Size	Size of the queue which stores the application objects processed by a thread pool.

- Stuck Thread Handling

Configures how to handle a thread when it is occupied by a specific task for longer than specified.

Item	Description
Max Stuck Thread Time	Length of time before a thread is identified as stuck.
Action On Stuck Thread	Action to take when stuck threads are detected. Choose one of the following: <ul style="list-style-type: none"> • None: Takes no action. • Interrupt: Sends an interrupt signal by calling <code>java.lang.Thread#interrupt()</code>. • IgnoreAndReplace: Ignores the Stuck Thread and replace it with a new thread. When the Stuck state is released, the ignored thread is discarded. • Warning: Leaves a thread dump with a warning in the log.
Stuck Thread Check Period	Interval in milliseconds between each consecutive check for the stuck thread status.
User Warning Class	If action-on-stuck-thread is set to Warning, the default value for this option is thread dump. However, you can write a class to execute the action you want by configuring this option. The class must implement the <code>jeus.util.pool.Warning</code> in <code>jclient.jar</code> . After writing the class, locate in <code>SERVER_HOME/lib/application</code> .

Application Example

The following is an example of an application using `ManagedExecutorService`.

Example of an Application using `ManagedExecutorService`

```
public class AppServlet extends HttpServlet implements Servlet {
    // Retrieve our executor instance.
    @Resource(name="mes1")
    ManagedExecutorService mes;

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        ArrayList<Callable> builderTasks = new ArrayList<Callable>();
        builderTasks.add(new AccountTask(reqID, accountID));
        builderTasks.add(new InsuranceTask(reqID, accountID));

        // Submit the tasks and wait.
        List<Future<Object>> results = mes.invokeAll(builderTasks);

        AccountInfo accountInfo = (AccountInfo) results.get(0).get();
        InsuranceInfo insInfo = (InsuranceInfo) results.get(1).get();
        // Process the results
    }
}
```

2.2. ManagedScheduledExecutorService

The `jakarta.enterprise.concurrent.ManagedScheduledExecutorService` interface inherits all the functions of `ManagedExecutorService`, as well as the functions of Java SE's `java.util.concurrent.ScheduledExecutorService`. This enables the interface to be able to execute tasks periodically/delays. In addition, tasks can be controlled by using the trigger and `ManagedTaskListener` interfaces.

Resource Definition Example

The following is an example of defining `ManagedScheduledExecutorService` as a resource.

Example of Defining `ManagedScheduledExecutorService` as a Resource: `<domain.xml>`

```
<domain>
  ...
  <server>
    <data-sources>
      <data-source>testdb</data-source>
    </data-sources>
    <managed-scheduled-executor-service>mes1</managed-scheduled-executor-service>
  </server>
  <resources>
    <managed-scheduled-executor-service>
      <export-name>mes1</export-name>
      <long-running-task>true</long-running-task>
      <thread-pool>
        <min>10</min>
```

```

<max>20</max>
<keep-alive-time>60000</keep-alive-time>
<queue-size>4096</queue-size>
<stuck-thread-handling>
  <max-stuck-thread-time>3600000</max-stuck-thread-time>
  <action-on-stuck-thread>None</action-on-stuck-thread>
  <stuck-thread-check-period>300000</stuck-thread-check-period>
</stuck-thread-handling>
</thread-pool>
</managed-scheduled-executor-service>
</resources>
...
</domain>

```

The configuration items are as follows.

- Basic Items

Item	Description
Export Name	Sets the name that will be used to register a managed scheduler executor service with the JNDI naming server.
Long Running Task	Indicates whether a task run by a managed scheduled executor service is long-running. Boolean type.

- Thread Pool

Configures the thread pool used in a managed executor service.

Item	Description
Min	Minimum number of threads managed by a thread pool.
Max	Maximum number of threads managed by a thread pool.
Keep Alive Time	Keep-alive time for inactive threads. If a thread pool contains more than the minimum number of threads, inactive threads for the specified time will be automatically removed. If this option is set to 0, threads will not be removed.
Queue Size	Size of the queue which stores the application objects processed by a thread pool.

- Stuck Thread Handling

Configures how to handle a thread when it is occupied by a specific task for longer than specified.

Item	Description
Max Stuck Thread Time	Length of time before a thread is identified as stuck.

Item	Description
Action On Stuck Thread	<p>Action to take when stuck threads are detected. Choose one of the following:</p> <ul style="list-style-type: none"> • None: Takes no action. • Interrupt: Sends an interrupt signal by calling <code>java.lang.Thread#interrupt()</code>. • IgnoreAndReplace: Ignores the Stuck Thread and replace it with a new thread. When the Stuck state is released, the ignored thread is discarded. • Warning: Leaves a thread dump with a warning in the log.
Stuck Thread Check Period	Interval in milliseconds between each consecutive check for the stuck thread status.
User Warning Class	If action-on-stuck-thread is set to Warning, the default value for this option is thread dump. However, you can write a class to execute the action you want by configuring this option. The class must implement the <code>jeus.util.pool.Warning</code> in <code>jclient.jar</code> . After writing the class, locate in <code>SERVER_HOME/lib/application</code> .

Application Example

The following is an example of an application using `ManagedScheduledExecutorService`.

Example of an Application using `ManagedScheduledExecutorService`

```
public class AppServlet extends HttpServlet implements Servlet {
    @Resource(name="mSES1")
    ManagedScheduledExecutorService mses;

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        Runnable printTask = new Runnable() {
            @Override
            public void run() {
                System.out.println(System.currentTimeMillis());
            }
        };
        // printTask is executed every 5 seconds
        mses.schedule(printTask, 5, TimeUnit.SECONDS);
    }
}
```

2.3. ContextService

The `ContextService` function provides a method for creating managed tasks instead of `ExecutorService`. By using the `ContextService` function, you do not have to worry about the context when creating a task, as the context is maintained when a task is executed in the application server.

Before executing a task by using a dynamic proxy, you configure the context in the thread, and then execute the task. Then, after all tasks have been performed, restore the context.

Resource Definition Example

The following is an example of defining ContextService as a resource.

Example of Defining ContextService as a Resource: <domain.xml>

```
<domain>
  ...
  <server>
    <data-sources>
      <data-source>testdb</data-source>
    </data-sources>
    <context-service>cs1</context-service>
  </server>

  <resources>
    <context-service>
      <export-name>cs1</export-name>
    </context-service>
  </resources>
  ...
</domain>
```

The configuration items are as follows.

- Basic Items

Item	Description
Export Name	Sets the name that will be used to register a context service with the JNDI naming server.

Application Example

The following is an example of an application using ContextService.

Example of an Application using ContextService

```
public class AppServlet extends HttpServlet implements Servlet {
    @Resource(name="cs1")
    ContextService cs;

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        // Regular Runnable task
        Runnable simpleTask = new Runnable() {
            @Override
            public void run() {
                int sum = 0;
                for (int i = 0; i < 10; i++) { sum += i; }
                System.out.println(sum);
            }
        };
    }
}
```

```

    }
};

// Convert a regular task to a contextual task through ContextService
cs.createContextualProxy(simpleTask, Runnable.class);

// Pass a contextual task to the Java SE executor
ExecutorService es = Executors.newFixedThreadPool(1);
es.submit(simpleTask);
}
}

```

2.4. ManagedThreadFactory

The `jakarta.enterprise.concurrent.ManagedThreadFactory` interface inherits Java SE's `java.util.concurrent.ThreadFactory` function, which enables it to create threads. Typically, it is used to submit `ThreadFactory` as the constructor's parameter when creating `ThreadPoolExecutor`. Therefore, even in Java SE's concurrency API, the context for a task can be maintained when a worker thread executes a task.

Resource Definition Example

The following is an example of defining `ManagedThreadFactory` as a resource.

Example of Defining `ManagedThreadFactory` as a Resource: `<domain.xml>`

```

<domain>
  ...
  <server>
    <data-sources>
      <data-source>testdb</data-source>
    </data-sources>
    <managed-thread-factory>mtf1</managed-thread-factory>
  </server>

  <resources>
    <managed-thread-factory>
      <export-name>mtf1</export-name>
      <thread-priority>5</thread-priority>
    </managed-thread-factory>
  </resources>
  ...
</domain>

```

The configuration items are as follows.

- Basic Items

Item	Description
Export Name	Sets the name that will be used to register a managed thread factory with the JNDI naming server.

Item	Description
Thread Priority	Sets thread priority. (Default value: 5)

Application Example

The following is an example of an application using `ManagedThreadFactory`.

Example of an Application using `ManagedThreadFactory`

```
public class AppServlet extends HttpServlet implements Servlet {
    // Retrieve our executor instance.
    @Resource(name="mtf1")
    ManagedThreadFactory mtf;

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        // Regular Runnable task
        Runnable simpleTask = new Runnable() {
            @Override
            public void run() {
                int sum = 0;
                for (int i = 0; i < 10; i++) { sum += i; }
                System.out.println(sum);
            }
        };

        // Execute simpleTask in a thread provided by ManagedThreadFactory
        mtf.newThread(simpleTask).start();

        // Or pass ThreadFactory as a parameter of ThreadPoolExecutor
        Executor e = new ThreadPoolExecutor(5, 10, 6L, TimeUnit.MINUTES,
            new ArrayBlockingQueue<Runnable>(4096), mtf);
        e.execute(new SimpleTask());
    }
}
```