Java EE 7 Recipes for Concurrency

Presented By: Josh Juneau
Author and Application Developer

About Me

Josh Juneau

Day Job: Developer and DBA @ Fermilab

Night/Weekend Job: Technical Writer

- Java Magazine and OTN
- Java EE 7 Recipes
- Introducing Java EE 7
- Java 8 Recipes

Twitter: @javajuneau

Agenda

Resolve a series of real life scenarios using the features of Concurrency Utilities for Java EE.

Java EE 7 Increases Productivity and Introduces Standards

Java EE 7 Increases Productivity

- CDI Everywhere
- JAX-RS Client API, Async Processing
- Bean Validation in EJBs and POJOs
- JSF Flows
- JMS 2.0

Java EE 7 Introduces Standards

- WebSockets
- JSON-P
- Batch API
- Concurrency Utilities for Java EE
- JPA Schema Generation

Concurrency Utilities for Java EE

Concurrency Utilities for Java EE - Now we have a standard for developing multi-threaded and concurrent applications

Recipes!

Problem #1

You are developing an online reservation system for a multi-million dollar company, and your client wishes to allow managers the ability to click a button to have the details of their all reservations sent to them, while the manager continues to navigate the site.

Solution

When the user clicks the button, send the task to a ManagedExecutorService on the application server for processing, and allow the user to continue their work.

Solution

```
@Resource(name = "concurrent/__defaultManagedExecutorService")
ManagedExecutorService mes;
```

```
AcmeReservationReport areport = new AcmeReservationReport("reservationReport",
                                    ejbFacade,
                                    reservationScheduleFacade,
                                    parkAdmissionFacade,
                                    parkFacade);
Future reportFuture = mes.submit(areport);
while (!reportFuture.isDone()) {
   // System.out.println("Running...");
}
if (reportFuture.isDone()) {
    System.out.println("Report Complete");
}
```

How it Works

- In Java SE 5, the java.util.concurrent package was introduced, providing a convenient and standard way to manage threads within Java SE applications
- The Concurrency Utilities for Java EE extends upon the SE implementation, making the concurrent resources injectable and placing them in the application server container
- All Java EE 7 Compliant Application Servers must contain default concurrent resources, in GlassFish these resources begin with _default, e.g: _defaultManagedExecutorService

How it Works

- Concurrency Utilities relies on JTA to maintain transaction boundaries
- User transaction support
- Manageable via JMX
- Access a Concurrent Resource via JNDI lookup or Injection:

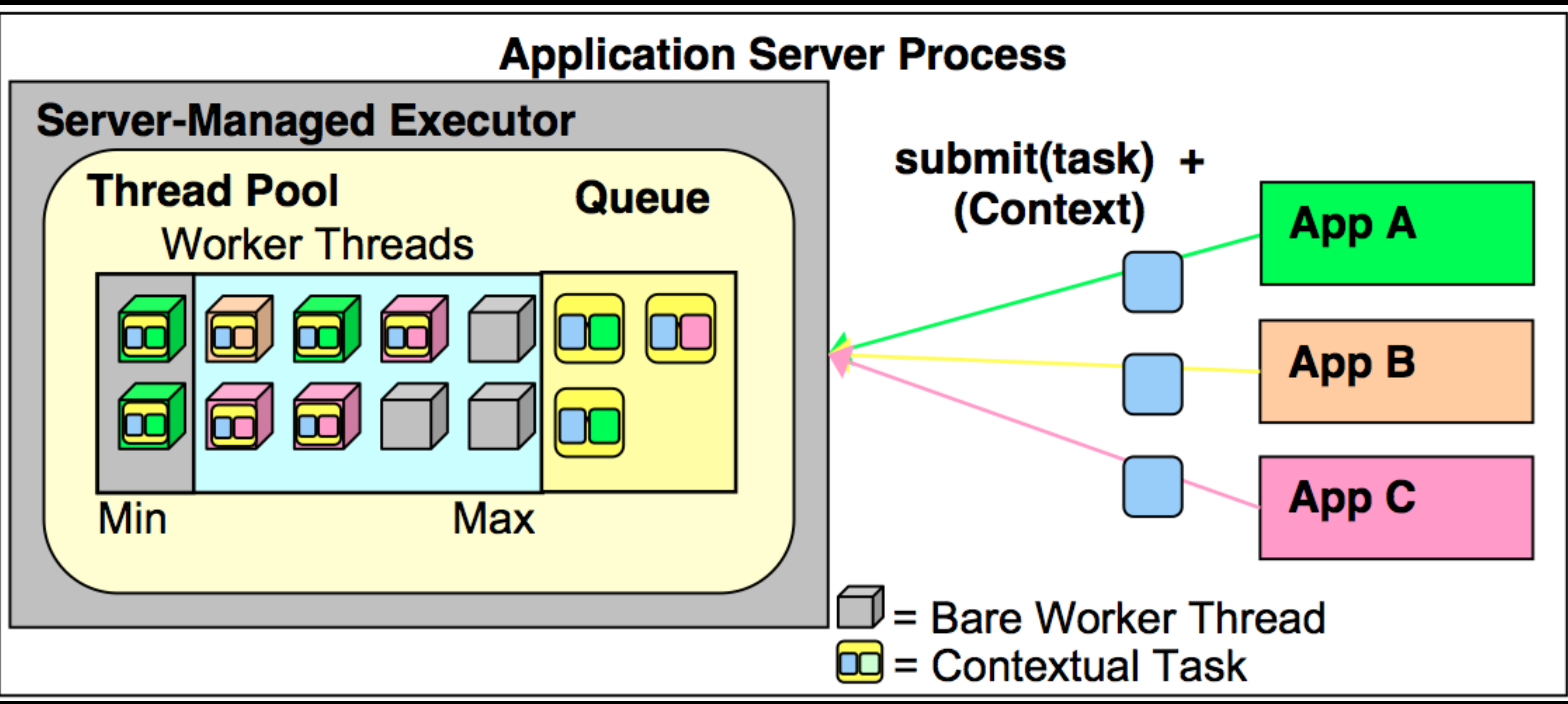
```
InitialContext ctx = new InitialContext();

ManagedExecutorService executor =
(ManagedExecutorService)ctx.lookup("java:comp/DefaultManagedExecutorService");
```

How it Works

- A "Task" is a unit of work that needs to be executed in a concurrent manner
    - *java.lang.Runnable or java.util.concurrent.Callable*, and optionally *ManagedTask*
- Context passed to the task
- To submit an individual task to a *ManagedExecutorService*, use the *submit* method to return a *Future* object
- To submit an individual task for execution at an arbitrary point, use the *execute* method

A Look at Lifecycle



Problem #2

You wish to invoke multiple long-running or resource intensive tasks in a concurrent manner

(Multiple reservation reports!)

Solution

Submit multiple tasks to a ManagedExecutorService by calling the *invokeAll* or *invokeAny* method

Solution

```java
Collection<Callable<ParkReservation>> parkReportList = new ArrayList<>();
for(int x = 0; x <= 3; x++){
    AcmeParkReservation res = new AcmeParkReservation(String.valueOf(x+1),
                                        new BigDecimal(x+1), ejbFacade);
    parkReportList.add(res);
}

try{
    System.out.println("Invoking reports...");
    // Return a List of Future objects
    reservations = mes.invokeAll(parkReportList);

    // Process each Future
    for(Future reservation:reservations){
        ParkReservation parkRes = (ParkReservation) reservation.get();
        //perform processing
        System.out.println("Processing Report: " + parkRes.getFirstName() + " " + parkRes.getLastName());
    }
```

```java
System.out.println("Invoking the reports...");
ParkReservation singleRes = mes.invokeAny(parkReportList);
```

How it Works

Works in the same manner as invocation of a single task, except pass a Collection of tasks:

- Call invokeAll to execute all tasks
- Call invokeAny to execute at most one

Problem #3

You wish to execute multiple background tasks, and then utilize the output of each together to formulate an end product.

Solution

Send each managed task to the ManagedExecutorService separately, and then work with each of the Future results once they are returned.

Solution

```
String reportId = "001";
BigDecimal accountID = new BigDecimal("1");
  Future<ParkReservation> parkReservationFuture = mes.submit(
          new AcmeParkReservation(reportId, accountID, parkReservationFacade));
  Future<RestaurantReservation> restaurantReservationFuture = mes.submit (
          new AcmeRestaurantReservation(reportId, accountID, restaurantReservationFacade));

  // Wait for the results.

  try {
    ParkReservation parkReservation = parkReservationFuture.get();
    RestaurantReservation restaurantReservation = restaurantReservationFuture.get();
  // Process the results
```

How it Works

It is possible to send multiple tasks to the ManagedExecutorService, and each of the tasks will return a separate Future object with which work can be completed.

Problem #4

You would like to schedule a task to be executed at a specified date and time.

Solution

Create a task class, implementing either Runnable or Callable, and pass it to the ManagedScheduledExecutorService

Solution

- Use a servlet to schedule tasks… AcmeWorldServletContextListener

```
Future counterHandle = null;
@Resource(name="concurrent/__defaultManagedScheduledExecutorService")
ManagedScheduledExecutorService mes;

@Override
public void contextInitialized(ServletContextEvent sce) {
    System.out.println("Context is initializing...");
    AcmeReservationCount reservationCount = new AcmeReservationCount();
     counterHandle = mes.scheduleAtFixedRate(
        reservationCount, 60, 60, TimeUnit.MINUTES);
}
```

How it Works

- Reference managed executor via JNDI lookup or injection via @Resource
- User transaction support

How it Works

- Invoke task using ManagedScheduledExecutorService by invoking scheduleAtFixedRate()
- Other options: schedule(), scheduleWithFixedDelay()
- Trigger API for further customization

Problem #5

You would like to fine tune a ScheduledManagedExecutorService to skip specified runs. In this case, we will skip every third run.

Solution

Utilize a Trigger class to implement the business logic to calculate which runs should be skipped, and pass the Trigger class to the ManageScheduledExecutorService with the Task.

How it Works

- Trigger API
  - Allows developers more control over how tasks are executed.
  - schedule(Runnable\Callable, Trigger)

Trigger Example

```
public class SkipSpecifiedTimeTrigger implements Trigger {

    Date scheduleDate;
    int skipTime;

    public SkipSpecifiedTimeTrigger(Date initial, int skipTime) {
        this.scheduleDate = initial;
        this.skipTime = skipTime;
    }

    @Override
    public Date getNextRunTime(LastExecution lastExecutionInfo, Date taskScheduledTime) {

        if (scheduleDate.before(taskScheduledTime)) {
            return null;
        }
        Date nextRunTime = scheduleDate;
        Calendar cal = Calendar.getInstance();
        cal.setTime(scheduleDate);
        cal.add(Calendar.SECOND, skipTime);
        scheduleDate = cal.getTime();
        return nextRunTime;
    }

    @Override
    public boolean skipRun(LastExecution lastExecutionInfo, Date scheduledRunTime) {
        return false;

    }
```

How it Works

- getNextRunTime(LastExecution, Date):
  - Retrieve the next date and time the task should run
  - Useful for gleaning information regarding the next task execution

How it Works

- skipRun(LastExecution, Date):
  - Returns a TRUE if the task run instance should be skipped
  - Useful for skipping task instances if the previous instance is paused or skipped
  - Once task is skipped, the Future state will throw a SkippedException…unchecked exceptions wrapped in SkippedException

Problem #6

You are interested in spawning a thread to periodically execute a task in the background

Solution

Spawn a server managed thread by passing a task via a ManagedThreadFactory

```
@Resource(name="concurrent/__defaultManagedThreadFactory")
ManagedThreadFactory threadFactory;
```

```
reservationReportMessage = "Starting alerter thread...";
Thread alerterThread = threadFactory.newThread(new ReservationAlerter());
System.out.println(alerterThread);
alerterThread.start();
```

How it Works

- A server managed Thread runs the same as any standard Thread...but in a managed fashion
- Utilize JNDI lookup or Injection via @Resource
- Context of invoking class can be passed to the Thread
- Threads can be managed via JMX
- User transaction support

Problem #7

You are interested in propagating contextual information from the Java EE container runtime to other threads.

Solution

Create a contextual proxy of a task that you wish to invoke later on.

```
@Resource
ContextService ctxSvc;
```

```
PromotionType newPromotion = new Promotion(name, authenticationController.getCurrentUser());

PromotionType proxy = ctxSvc.createContextualProxy(newPromotion, PromotionType.class);
promotionController.addPromotion(proxy);
```

In another session…

```
<p:dataTable value="#{promotionController.obtainPromotions()}" var="promotion">
    <p:column headerText="Task">
        <p:commandButton action="#{promotion.processPromotion()}" value="Process Promotion"
                         update="@form"/>

    </p:column>
```

How it Works

- Context of application is captured upon creation of proxy
- Proxy object methods can be run in the captured context at a later time

How it Works

- Creating Proxy:
  - createContextualProxy - various implementations
- Utilizes java.lang.reflect package

How it Works

- Dynamic proxy can be customized via execution properties
- Return execution properties of given instance:
  - Map<String, String> getExecutionProperties(Object contextualProxy)

How it Works

Can be executed on same transaction context a invoking thread:

- Execution Task: ManagedTask.TRANSACTION

Use Cases:

- Propagate Security identity
- Run tasks within a different context

Problem #8

You would like to monitor the lifecycle current asynchronous operations that are running within your Java EE Container.

Solution

Utilize a ManagedTaskListener to manage your Task's lifecycle.

How it Works

- Allow managed task class to implement ManagedTask interface

OR

- ManagedTaskListener is submitted along with the Task class to the executor via the ManagedExecutors.managedTask utility method

How it Works

- Create a custom class that implements ManagedTaskListener, and override methods accordingly

```
public class CustomManagedTaskListener implements ManagedTaskListener {

    @Override
    public void taskSubmitted(Future<?> future, ManagedExecutorService mes, Object o) {
        System.out.println("Task Submitted");
    }

    @Override
    public void taskAborted(Future<?> future, ManagedExecutorService mes, Object o, Throwable thrwb
        System.out.println("Task Aborted");
    }

    @Override
```

How it Works

- Lifecycle Monitoring:
  - taskSubmitted
  - taskAborted
  - taskStarting
  - taskDone

Problem #9

Your application consists of a JSF view containing a tabbed UI. You'd like to load data on different tabs using separate concurrent tasks.

Solution

Utilize a separate Managed ExecutorService for initiating tasks in each of the tabs. Utilize a sophisticated user interface that informs the user of the current status on each tab.

How it Works

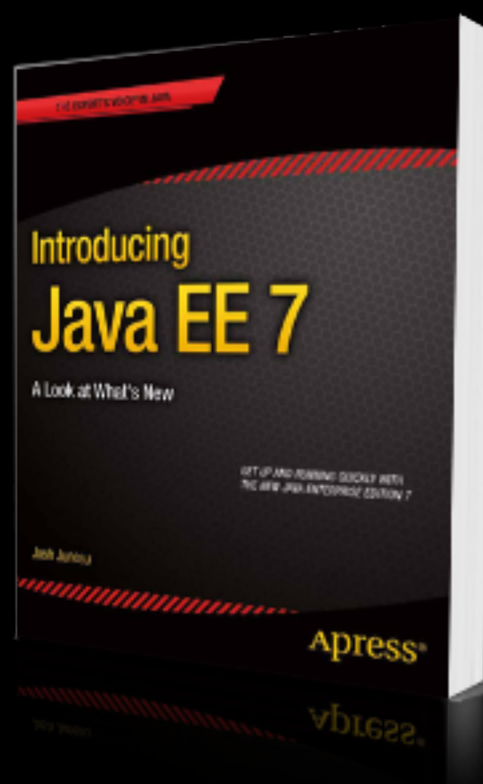
Separate ManagedExecutorServices initiated from each different tab.

Utilization of PrimeFaces poll component to periodically check upon the status of the Future objects, and supply informative feedback to the user.

Problem #10

You would like to learn more about Concurrency Utilities for Java EE…

Learn More



Code Examples: https://github.com/juneau001/AcmeWorld

Contact on Twitter: @javajuneau

Contact

Josh Juneau

- Java EE 7 Recipes
- Introducing Java EE 7
- Java 8 Recipes

Twitter: @javajuneau