Concurrency in Enterprise Java

Alexander Heusingfeld - @goldstift

Twitter: #javaland #JSR236

JavaLand 2014

We take care of it - personally!

innoQ

There ain’t no fuzz about this, right?

innoQ

```
Runnable myRunnable = new Runnable() {
    public void run() {
        // do stuff in Thread
    }
};
Thread myThread = new Thread(myRunnable);
myThread.start(); // ...and off we go!
```

http://www.flickr.com/photos/girlaphid/3818224024/sizes/l/

New Thread considered dangerous in EE

1. No management of Thread lifecycle
2. No management of number of Threads
3. Monitoring tedious/ almost impossible
4. No propagation of context

http://www.flickr.com/photos/spanishflea/215064063/sizes/o/

java.util.concurrent

innoQ

ExecutorServices

```
Executors.newSingleThreadExecutor()

Executors.newCachedThreadPool()

Executors.newFixedThreadPool()

Executors.newScheduledThreadPool()
```

JSR236

javax.enterprise.concurrent

What’s in it for the devs?

innoQ

ManagedThreadFactory

ManagedThreadFactory

- extends the Java SE ThreadFactory
- same API as `Thread.newThread(Runnable)`
- Threads returned implement `ManagableThread`
- Enables use of Java SE concurrency utilities like `Executors`

ExecutorServices for Java EE

```
Executors.newSingleThreadExecutor(ThreadFactory)

Executors.newCachedThreadPool(ThreadFactory)

Executors.newFixedThreadPool(int, ThreadFactory)

Executors.newScheduledThreadPool(int, ThreadFactory)
```


innoQ

```
@Resource ManagedThreadFactory factory;

protected void processRequest(ServiceRequestData data) {

    // Creating Java SE style ExecutorService
    ExecutorService executor = Executors.newFixedThreadPool(10, factory);
    Future f = executor.submit(new MyTask(data));
}
```

```
 1 public class MyTask implements Runnable {
 2     public void run() {
 3         InitialContext ctx = null;
 4         try {
 5             ctx = new InitialContext();
 6             UserTransaction ut = (UserTransaction) ctx.lookup(
 7                     "java:comp/UserTransaction");
 8             ut.begin();
 9             while (!((ManageableThread) Thread.currentThread()).isShutdown()){
10
11                 // Perform transactional business logic?
12             }
13             // do cleanup
14             ut.commit();
15         } catch (Exception ex) {
16             ex.printStackTrace();
17         }
18     }
19 }
```

ManagedExecutorService

ManagedExecutorService

- extends the Java SE ExecutorService
- methods for submitting tasks to Java EE environment
- available via JNDI look-up or `@Resource` injection
- Tasks must implement `Runnable` or `Callable`
- Lifecycle APIs disabled -> throw `IllegalStateException`

innoQ

```java
@Resource ManagedExecutorService mes;

protected String processRequest(ServiceRequest request) {
    Future<String> f = mes.submit(new Callable<String>() {
       public String call() {
           // do something with context
           return "result";
       }
    });
    try {
        return f.get();
    } catch (InterruptedException | ExecutionException ex) {
        throw new IllegalStateException("Cannot get the answer", ex);
    }
}
```


innoQ

```
1  @Resource ManagedExecutorService mes;
2
3  class MyLongReturningTask implements Callable<Long> { . . . }
4  class MyDateReturningTask implements Callable<Date> { . . . }
5
6  protected void processRequest(ServiceRequest request) throws
7    ExecutionException {
8
9     ArrayList<Callable> tasks = new ArrayList<();
10    tasks.add(new MyLongReturningTask());
11    tasks.add(new MyDateReturningTask());
12    List<Future<Object>> result = executor.invokeAll(tasks);
13 }
```

```
 1 @WebServlet(urlPatterns = "/MyAsyncServlet", asyncSupported = true)
 2 public class MyAsyncServlet extends HttpServlet {
 3
 4     @Resource ManagedExecutorService executor;
 5
 6     protected void doGet(HttpServletRequest request, HttpServletResponse
 7                          response) throws ServletException, IOException {
 8
 9         final AsyncContext ac = request.startAsync();
10         Runnable task = new Runnable() {
11             public void run() {
12                 // ...
13                 ac.complete();
14             }
15         }
16         executor.submit(task);
17     }
18 }
```


innoQ

ManagedScheduledExecutorService

ManagedScheduledExecutorService

- extends ManagedExecutorService
- extends ScheduledExecutorService
- additional methods to schedule tasks with new `Trigger`
- new methods also return `ScheduledFuture` objects

innoQ

```
@Resource ManagedScheduledExecutorService executor;

protected void processRequest(ServiceRequest request) {
    ScheduledFuture<?> f = executor.scheduleAtFixedRate(
        new MyRunnableTask(request), 5, 10, TimeUnit.SECONDS);

}
```

```java
public class MyTrigger implements Trigger {

    private final Date firetime;

    public MyTrigger(Date firetime) { this.firetime = firetime; }

    @Override
    public Date getNextRunTime(LastExecution le, Date taskScheduledTime) {
        if (firetime.before(taskScheduledTime)) {
            return null;
        }
        return firetime;
    }

    @Override
    public boolean skipRun(LastExecution le, Date scheduledRunTime) {
        return firetime.before(scheduledRunTime);
    }
}
```


innoQ

ContextService

ContextService

- creating contextual proxies
- Examples of context: classloading, namespace, security
- customisable through ExecutionProperties
- Can be run on transaction context of invoking thread (if any)

```
1 // within servlet or EJB method
2 @Resource ContextService service;
3
4 // Capture application context for later execution
5 IMessageProcessor processor = ...;
6 IMyAppWorker proxy = service.createContextualProxy(processor,
7                      IMyAppWorker.class);
8 cache.put(IMyAppWorker.class, proxy);
```

```
12 // at a later time in a different thread retrieve the proxy from the cache
13 IMyAppWorker worker = cache.get(IMyAppWorker.class);
14 // and execute with the context of the servlet or EJB
15 worker.processMessage(msg);
```

Convenience #1 - ManagedTaskListener

```
void taskSubmitted(java.util.concurrent.Future<?> future,
                   ManagedExecutorService executor, Object task)

void taskStarting(java.util.concurrent.Future<?> future,
                  ManagedExecutorService executor, Object task)

void taskAborted(java.util.concurrent.Future<?> future,
                 ManagedExecutorService executor, Object task, java.lang.
                 Throwable exception)

void taskDone(java.util.concurrent.Future<?> future, ManagedExecutorService
              executor, Object task, java.lang.Throwable exception)
```


innoQ

Convenience #2 - ManagedExecutors

```
// Register listener for task with specific executionProperties.
// Adopts executionProperties of ManageableTask
Runnable managedTask(Runnable, ManagedTaskListener)
Runnable managedTask(Runnable, Map<String, String>,
                               ManagedTaskListener)
Callable managedTask(Callable, ManagedTaskListener)
Callable managedTask(Callable, Map<String, String>,
                               ManagedTaskListener)

// Checks currentThread instanceof ManageableThread
ManagedExecutors.isCurrentThreadShutdown()
```


innoQ

DevOps? How do we do it?

innoQ

Setup - CLI

WildFly CLI

```
$JBOSS_HOME/bin/jboss-cli.sh –connect “/subsystem=ee/managed-
executor-service=async-http-mes/:add(
 core-threads=20,
 jndi-name=java:comp/async-http-mes,
 context-service=default,
 hung-task-threshold=110000,
 keepalive-time=60000,
 thread-factory=default,
 queue-length=50,
 reject-policy=ABORT,
 max-threads=500)”
```

WildFly CLI - Monitoring

```
$JBOSS_HOME/bin/jboss-cli.sh --connect "/subsystem=ee/managed-executor-
service=async-http-mes/:read-resource(recursive-depth=0,include-
runtime=true)"
{
    "outcome" => "success",
    "result" => {
        "context-service" => "default",
        "core-threads" => 20,
        "hung-task-threshold" => 110000L,
        "jndi-name" => "java:jboss/ee/concurrency/executor/async-http-mes",
        "keepalive-time" => 60000L,
        "long-running-tasks" => false,
        "max-threads" => 500,
        "queue-length" => 50,
        "reject-policy" => "ABORT",
        "thread-factory" => "default"
    }
}
```

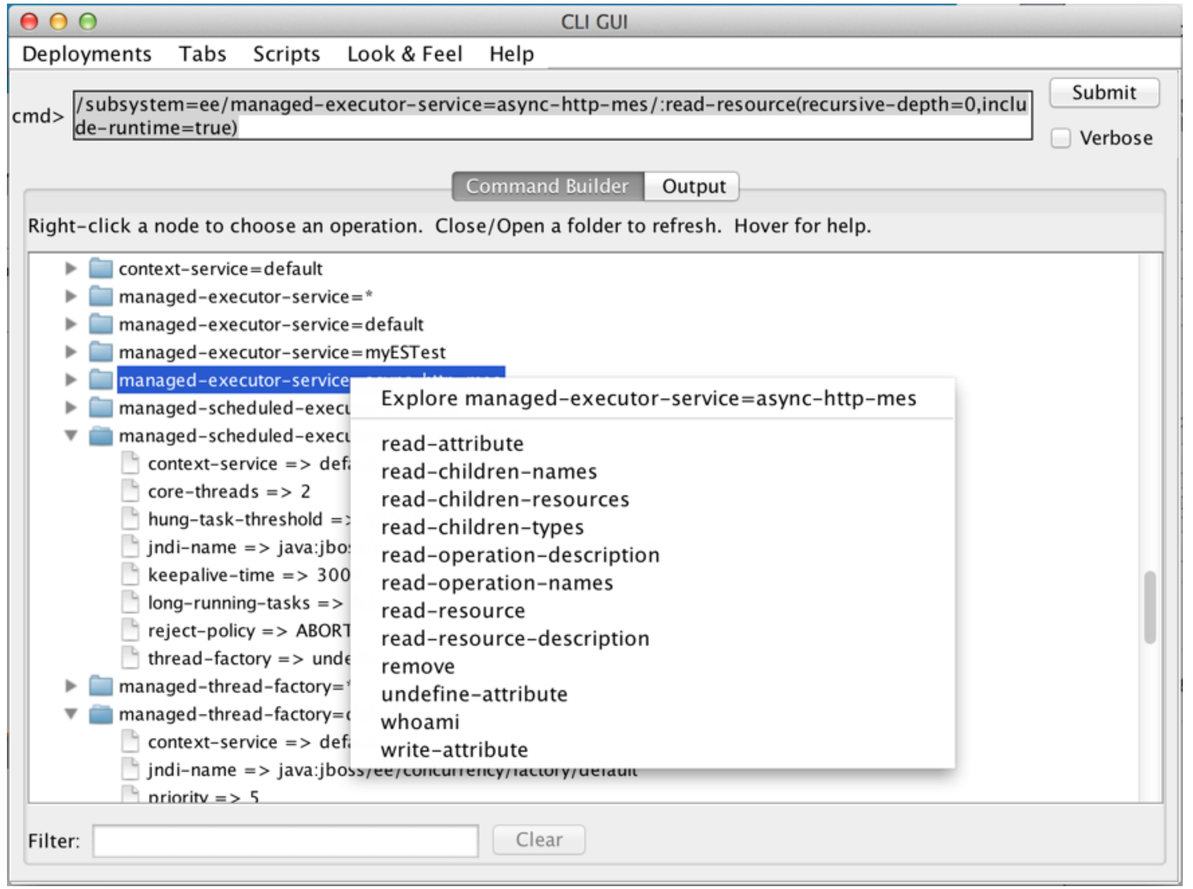

innoQ

Too complicated?

innoQ

Wildfly CLI command builder

```
$JBOSS_HOME/bin/jboss-cli.sh --gui
```



Still too complicated?

innoQ

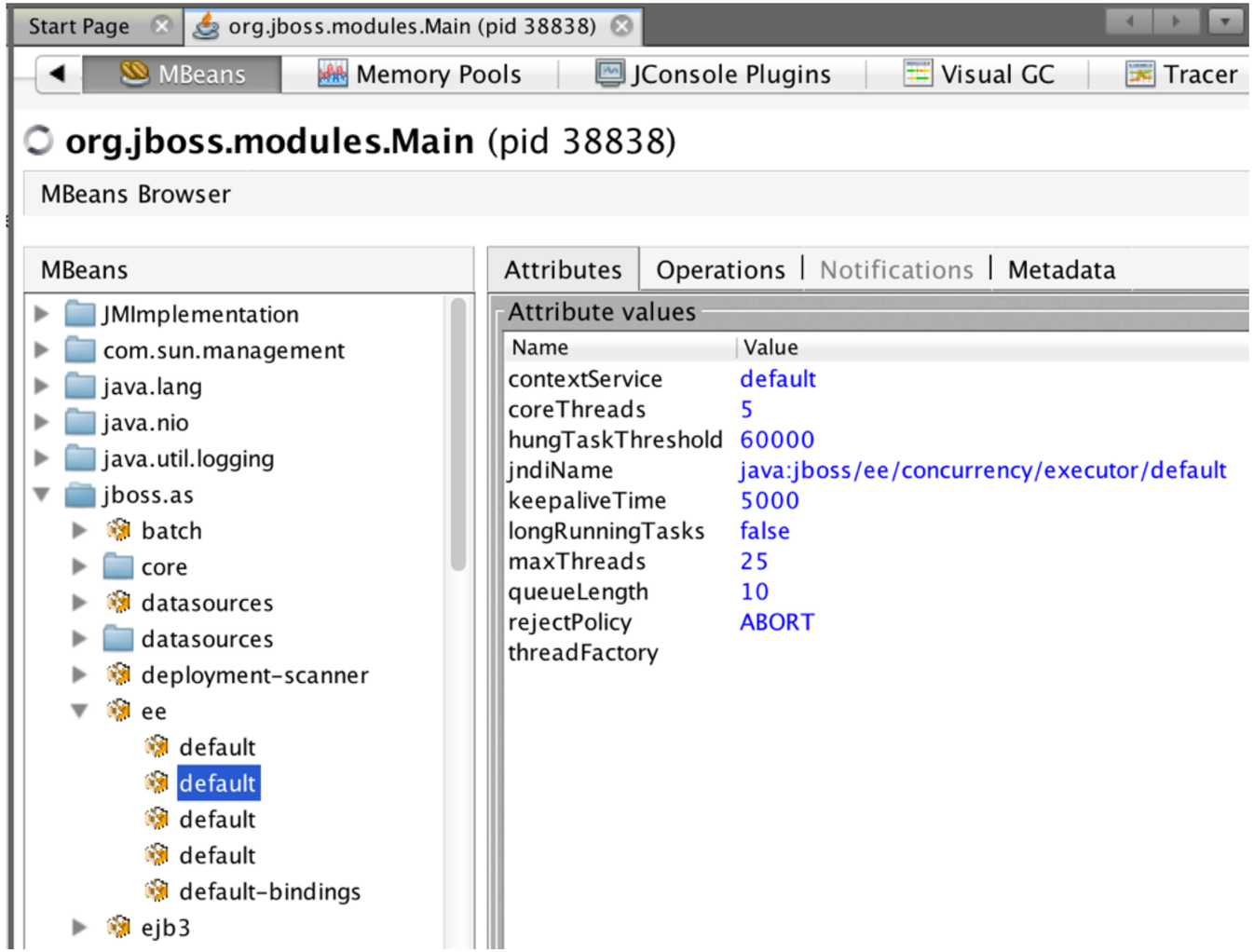
Management & Monitoring via JMX

Everything published as MBeans

-> values

-> operations

-> metadata



innoQ

Everything fine?

innoQ

From http://www.flickr.com/photos/upturnedface/2700127506/sizes/o/

1. The single ExecutorService

```
1 @Resource ManagedExecutorService mes;
2
3 protected void processRequest(ServiceRequest request) throws
4   ExecutionException {
5         Future<String> f = mes.submit(new Callable<String>() {...});
6         System.out.println(â€œCallable result: " + f.get());
7 }
```

2. The queue that queued

```
@Resource ManagedThreadFactory factory;

protected void processRequest(ServiceRequestData data) {

    // Creating Java SE style ExecutorService
    BlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>(10);
    ThreadPoolExecutor executor = new ThreadPoolExecutor(5, 10,
                                   0L, TimeUnit.MILLISECONDS,
                                  queue,
                                   factory);
    // MyRunnable implements .equals() and .hashcode()
    MyRunnable task = new MyRunnable(data);

    // prevent duplicate queue entries
    if (!queue.contains(task)) {
        executor.submit(task);
    }
}
```

3. A self overtaking job

```
@Resource ManagedScheduledExecutorService executor;

protected void processRequest(ServiceRequest job) {
    ScheduledFuture<?> f = executor.scheduleAtFixedRate(
        new MyRunnableTask(job), 5, 10, TimeUnit.SECONDS);

}
```


innoQ

4. Know your RejectionPolicy

```
<bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="maxPoolSize" value="5"/>
    <property name="queueCapacity" value="180"/>
    <property name="rejectedExecutionHandler">
        <bean class="java.util.concurrent.ThreadPoolExecutor$DiscardPolicy"/>
    </property>
</bean>
```

http://www.flickr.com/photos/weaselmcfee/3721267658/sizes/o/

Spring's JSR236 Support

Provides configurability & monitoring via JSR236

-› ConcurrentTaskExecutor auto-detects JSR236 classes

-› Uses specified ManagedExecutor transparently

-› Check the code: Github commit 4004e53

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

-› Streams

-› CompletableFuture

Summary

-› good mix of convenience & flexibility

-› doesn’t reinvent the wheel

-› potential for optimisation:

  -› API to create ManagedExecutorService

  -› validation/ constraints for ManagedExecutorService

innoQ

Links

- JSR236 spec: https://jcp.org/en/jsr/detail?id=236
- Java EE 7 JavaDoc: http://docs.oracle.com/javaee/7/api/
- Java EE 7 samples: https://github.com/javaee-samples/javaee7-samples
- Wildfly docs: https://docs.jboss.org/author/display/WFLY8/EE+Subsystem+Configuration

innoQ

Thanks for your attention!

Alexander Heusingfeld, **@goldstift**

alexander.heusingfeld@innoq.com

http://www.innoq.com/de/timeline

Twitter: #javaland #jsr236

We take care of it - personally!

innoQ